Tamb-ON: Juego de ritmo con creación de nivel basada en MIDI
Tamb-ON: MIDI-based rhythm game with level design

Curso 2020-2021, convocatoria de junio

Facultad de Informática

Grado de Ingeniería Informática y Grado en Desarrollo de Videojuegos

Trabajo de Fin de Grado. Universidad Complutense de Madrid

Guillermo Alonso Patiño
Ignacio de la Cruz Crespo
Belén Solla Sanz

Director

Adrian Riesco Rodriguez

Resumen

Tamb-On es un juego de ritmo que representa las notas de las canciones con círculos de colores. Cada círculo simboliza una nota, de forma similar a cómo se representan en el juego “Taiko no Tatsujin”. La característica principal del juego es que permite la integración de niveles basados en MIDI, o lo que es lo mismo, que un jugador puede introducir sus canciones favoritas y jugarlas.

En esta memoria se habla sobre el TFG de “Tamb-on”, explicando cómo se desarrolló el juego, las dificultades encontradas, qué se aprendió en este desarrollo, pues algunos integrantes eran novatos en la creación de videojuegos. La memoria finaliza con unas conclusiones, además de una discusión sobre posibles mejoras en el futuro.

Palabras clave:

MIDI, Tamb-ON!, Taiko, Videojuego, Ritmo, Unity

Summary

Tamb-On is a rhythm game that depicts the songs' notes with colored circles. Each circle represents one note, in a similar way to how they are represented in the game "Taiko no Tatsujin". The main feature of the game is that it allows the integration of MIDI-based levels, meaning that the player can set up their favorite songs and play them in game.

This paper centers around Bachelor's Thesis Project "Tamb-on", explaining how the game was developed, the struggles that had to be overcome, the learning process as well as personal thoughts and opinions. The paper closes with a section for conclusions and suggestions for future improvements.

Key words:

MIDI, Tamb-ON!, Taiko, Video Game, Rhythm, Unity

1. Introducción

En este Trabajo de Fin de Grado (TFG) se desarrolla un videojuego llamado **Tamb-On**, inspirado en el juego *Taiko no tatsujin*, un juego de ritmo donde se reta al jugador a mantener el ritmo de la canción. El equipo está compuesto por 2 alumnos del Grado de Ingeniería Informática y 1 alumna del Grado de Desarrollo de Videojuegos.

La idea original es crear un juego rítmico, divertido, preferiblemente usando Unity como se explicará más adelante. Para entender mejor qué estamos exponiendo en este trabajo necesitamos ver el juego en el que nos hemos inspirado.

1.1 Taiko no tatsujin

Figura 1: un tambor tradicional taiko

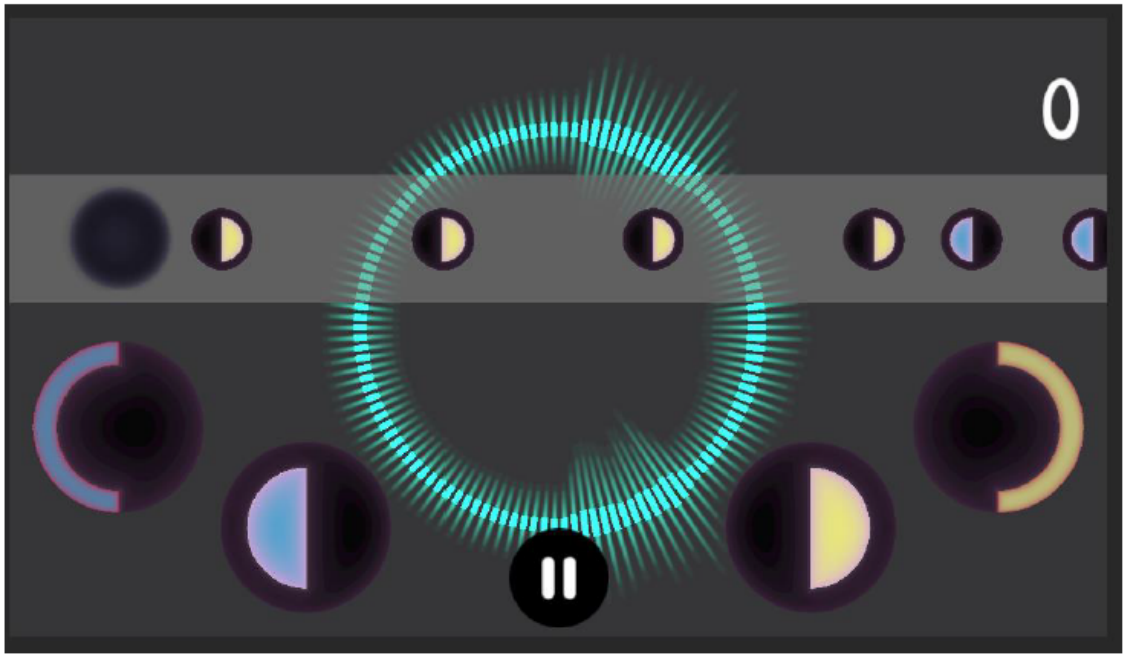
Taiko no tatsujin es un juego de ritmo donde el jugador debe acertar unas notas de golpe, llamadas "notas" de aquí en adelante, que representan los sonidos de una canción. Aparecen siguiendo el ritmo de la melodía y el jugador debe pulsar tantas como pueda para conseguir la máxima puntuación posible. Este juego tiene una estética que representa la cultura japonesa. De hecho, el instrumento representado en el juego es un Taiko, un instrumento japonés similar a un tambor, como se muestra en la figura 1. Fuente de la imagen.

El código del proyecto lo podemos encontrar en Github y la apk la encontramos aquí.

1.2 Tamb-On

Tamb-On es el juego desarrollado en este TFG y está inspirado en *Taiko no Tatsujin* pero con diferencias en sus mecánicas, dinámicas y estética. La temática elegida deja atrás el concepto tradicional japonés y apuesta por un estilo más electrónico y moderno. Las notas musicales están representadas con la misma textura que el botón que el usuario tiene que pulsar (véase figura 2) para simplificar su uso y hacerlo más intuitivo.

Figura 2: vista de la pantalla de juego



Dentro de **Tamb-On** existe un conjunto de dificultades, mutadores (ver figura 4) y personajes (ver figura 3) que cuando se combinan entre ellos permiten la personalización del juego a gusto del jugador. La dificultad permite configurar el desafío que supone cada canción. Los mutadores y los personajes modifican la experiencia de cada partida, añadiendo o suprimiendo mecánicas de juego para cada nivel. Así permitimos que una misma canción tenga una sensación diferente en cada juego si se cambia de personaje o mutadores, aumentando la rejugabilidad de cada canción y reduciendo la sensación de repetición, logrando así que el juego sea más divertido.

Figura 3: pantalla de selección de personajes

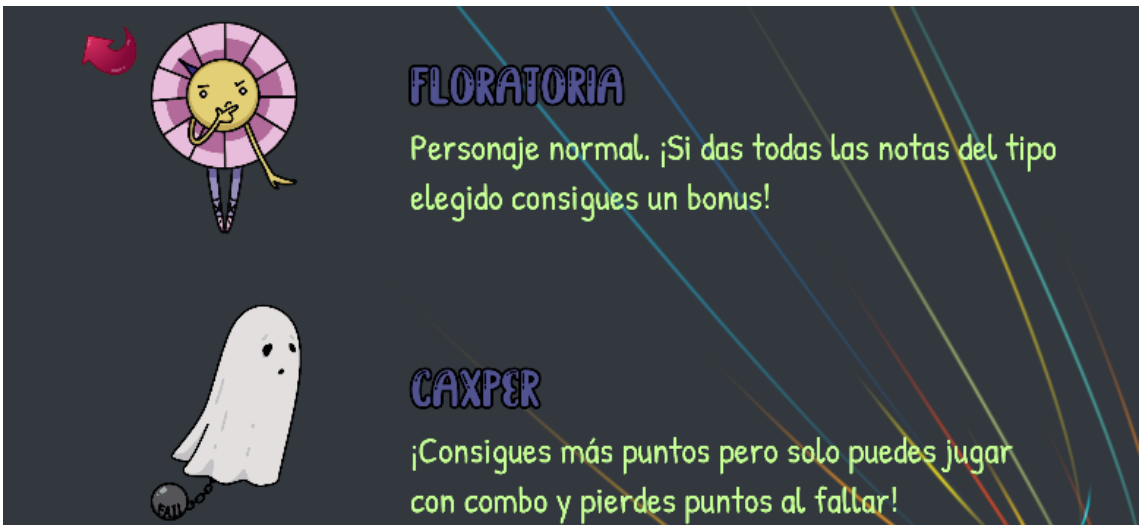


Figura 4: pantalla de selección de mutadores

1.3 Características implementadas

A continuación se describen una serie de características destacables dentro de nuestra implementación. Se incluyen las que diferencian al juego de otros dentro del mismo género:

- Una versión para Android del juego y otra versión de PC. Sin embargo esta versión es un traslado de la versión para móviles.
- Un sistema de combos y puntuación dinámica, donde las notas valen más o menos en función de las notas acertadas consecutivamente.
- Mecánicas para los personajes, aportandoles dinámicas menos planas. Los personajes tienen efectos perceptibles a lo largo de toda la partida, como un

sistema de combos propio en el caso de Caxper, un modo de juego más relajado para el caso de Topita o como hace su antítesis, Silvestre, un personaje que solo se le permite hacer “perfectos”.
- Mecánicas para los mutadores. Estas se pueden permutar entre sí y combinar con un personaje, permitiendo al usuario obtener una experiencia más personal y acorde a sus gustos. Por ejemplo, un modo “ciego” que reduce el campo de visión del usuario. Estos mutadores afectan al sistema de puntuación, recompensando o perjudicando a los jugadores según su configuración.
- Un sistema de amigos donde los jugadores podrán ver qué amigos están en línea y qué canciones están jugando.
- Una tabla de récords por canciones que muestran los récords personales y mundiales. Estas tablas muestran la canción, el jugador, el personaje y los mutadores elegidos así como la puntuación final obtenida.
- Creación de niveles de juego usando archivos MIDI.

1.4. Plan de trabajo

Aquí explicaremos brevemente nuestro plan de trabajo, mostrando las ideas originales y los cambios ocurridos en dichas ideas. También introduciremos de forma breve el desarrollo de **Tamb-On** sin profundizar en detalles técnicos.

1.4.1 Presentación del equipo y roles

El equipo desarrollador está compuesto por Ignacio de Cruz Crespo, aportando conocimientos de teoría musical y de informática musical para el uso del audio y del ritmo en este proyecto junto con un rol en la programación interna, Belén Solla Sanz, con roles multidisciplinares tanto en programación y diseño como en arte, y Guillermo Alonso Patiño, proporcionando conocimientos de diseño de interfaces, experiencia en desarrollo con lenguajes similares a C# y manteniendo un rol de documentador.

1.4.2 Objetivos del proyecto

El objetivo final de este TFG es crear un videojuego con una identidad lo suficientemente definida como para diferenciarse de otros juegos de ritmo, entre ellos *Taiko no Tatsujin*, y que permita el uso de archivos MIDI para la creación de niveles. El juego debe tener una versión compatible con móviles, en concreto una versión para Android, y realizar una versión para PC más adelante. El juego se basará en unas mecánicas que proporcionan al usuario tanto una configuración acorde a sus gustos como una interacción divertida con este. Teniendo en cuenta que el juego puede volverse tedioso de rejugar, se busca crear dinámicas que lidien con este problema y que le den un aire menos plano o estático, haciendo que sea menos aburrido en ciertos puntos por lo simple de sus mecánicas.

Aparte de los objetivos principales del proyecto, los ingenieros informáticos querían aprender cómo utilizar Unity, el motor elegido para el proyecto. Esto es

más un objetivo personal que un objetivo del proyecto. Nunca habían desarrollado un videojuego y este proyecto brindaba la oportunidad.

1.4.3 Organización de la memoria

El resto de la memoria se organiza por secciones que cubren distintos aspectos del proceso de desarrollo. En la sección 3, "*Preliminares*", se presenta la planificación del trabajo así como las herramientas utilizadas. En la sección 4, "*Conceptos Básicos del Juego*", se desarrollan las mecánicas y dinámicas, explicándolas con detalle desde el punto de vista de la experiencia de usuario. También se detalla un ciclo de juego típico, exponiendo de esta manera la interacción de mecánicas en el contexto de una partida. La sección 5, "*Fase de Desarrollo*", ahonda más en los aspectos técnicos de las mecánicas para, junto con la explicación orientada al diseño de videojuegos de la sección anterior, proporcionar una visión completa de los elementos que componen el juego. Se mencionan tanto partes relativas a la programación como al diseño, al arte y las redes. En la sección 6, "*Fases de Prueba*", se presentan los resultados obtenidos en las pruebas con usuarios, y cómo actuamos ante este feedback. La sección 7, "Trabajo Personal", se centra en la experiencia personal de los integrantes del equipo y cómo se consiguió hacer uso de los puntos fuertes de cada uno. Finalmente, las secciones 8 y 9, "*Conclusiones y Trabajo a Futuro*" actúan de cierre al recapitular todo lo expuesto anteriormente y sugerir mejoras mirando al futuro.

2. Introduction

This Bachelor's Thesis Project, or TFG in Spanish, centers around the development of a video game called **Tamb-On**, which is inspired by the game *Taiko no Tatsujin*. This game's goal is to challenge the player to keep the beat of the song. The team that developed this project was made up of two people studying a *Grado de Ingeniería Informática* (Bachelor in Computer Science Engineering) and one studying a *Grado en Desarrollo de Videojuegos* (Bachelor in Video Game Development).

The original goal was to create a fun, rhythmic game, using Unity Engine if possible, as explained later in the paper. In order to understand the thoughts behind the project better, and to give some context, we need to expand on the concept of the game that inspired the thesis.

2.1 Taiko no tatsujin

Figure 1: a traditional taiko drum

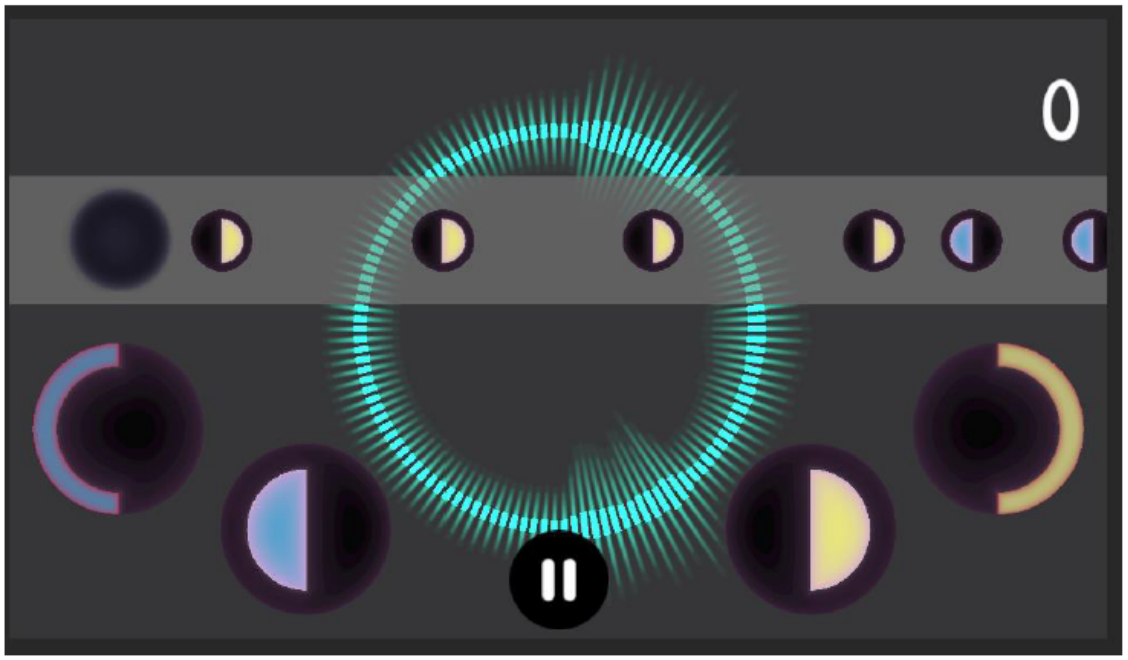
Taiko no Tatsujin is a rhythm game in which the player hits "beat notes" (from now on called beats) in sync with a song. These beats represent strong rhythmical notes in the song. The player has to hit as many beats as possible to get the best score they can. The game is heavily based on a Japanese aesthetic and uses elements from its culture and tradition. In fact, the instrument depicted by the game is a taiko drum, a Japanese percussion instrument (see Figure 1). Image source.

El código del proyecto lo podemos encontrar en: github, la apk la encontramos en aquí.

2.2 Tamb-On

The game developed for this thesis is **Tamb-On**, which was inspired by Taiko no Tatsujin but with noticeable differences in mechanics, dynamics and artistic choices. Tamb-On leaves the Japanese aesthetic behind, and instead sports a more modern, electronic look. Song beats are represented with the same texture as the buttons the player uses for game input (see Figure 2). This is a deliberate choice that aims for easiness of use and intuitive playing.

Figure 2: screenshot of the gameplay



Tamb-On offers the player a customizable game experience thanks to its varying levels of difficulty along with the mutator (see Figure 4) and character systems (see Figure 3). The combination of these allows for a unique player experience. Difficulty level lets the player choose how challenging the game will be. Mutators and characters add a layer to each playthrough by turning certain game mechanics on or off. All of these factors combined make the game feel different for each configuration, which ensures replayability and keeps the game fun for a longer period of time.

Figure 3: screenshot of the character selection menu

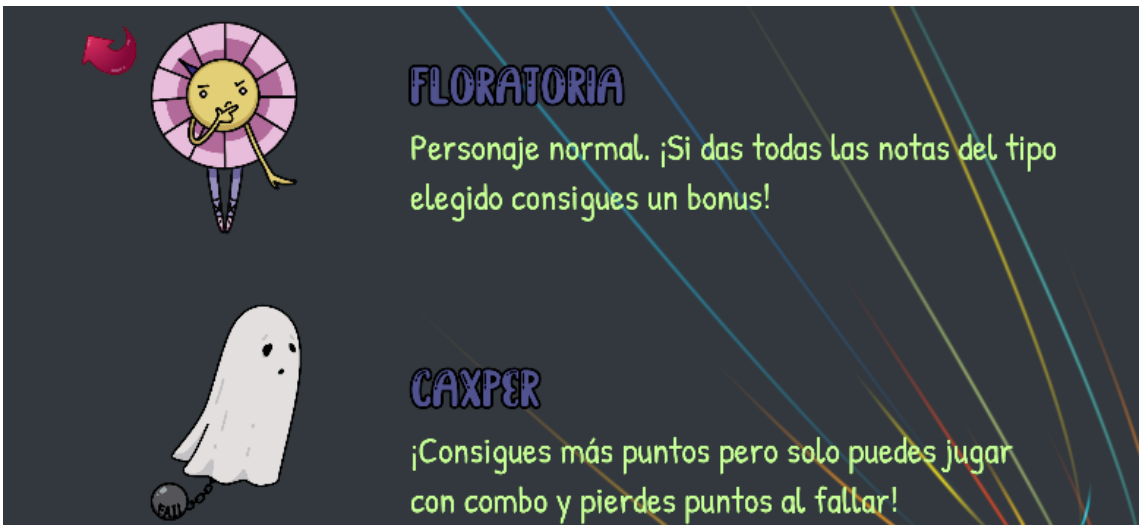


Figure 4: screenshot of the mutator selection menu

2.3 Implemented features

Here is a comprehensive list of the key features implemented in our game. It also includes what makes our features distinct from other games within the rhythm genre:

- One version for Android devices and one for PC. However, the PC version is a port of the Android version.
- Dynamic point and combo systems, which are influenced by the amount of beats hit in succession.
- Character mechanics, contributing to more complex game dynamics. Characters have palpable effects all along the game, for example a unique combo system for Caxper, a more relaxed game experience with Topita or, in a completely opposite way, a challenging character like Silvestre who can only count perfect hits.
- Mutator mechanics. They can be switched on and off and combined with a character, allowing the player to get a personalized game experience, according with their own individual preferences. One example would be the "blind" mode, which decreases vision range for the player by spawning beats closer to the goal. Mutators influence the point system, rewarding or punishing players depending on their chosen configuration.
- A friend hub system that allows players to add friends, and see which ones are online or which songs they are playing.
- A section for best scores, with one best scores table for each song. They show the song name, player name, character and mutators active for that song, as well as final score.
- Level creation using MIDI files.

2.4. Work Plan

This section goes over our work plan, explaining some of our original ideas and how they changed over time. It contains a brief introduction of Tamb-On's development as well, glossing over technical feats.

2.4.1 Team introduction and roles

The developer team consists of: Ignacio de Cruz Crespo, contributing with music theory and music computing knowledge, used in the project's audio and rhythm features, as well as programming. Belén Solla Sanz, with multidisciplinary roles ranging from programming, game design and art. Guillermo Alonso Patiño, contributing with UI design, previous knowledge with similar programming languages to C# and keeping a documenting role.

2.4.2 Project goals

The final goal of this project is to develop a video game with a distinct identity, different enough from other rhythm games like *Taiko no Tatsujin* and with a level

system that allows the use of MIDI files for their creation. On top of that, an online social/friend feature; all of these features combined on an Android based app which could be ported to PC in future iterations of the project. The game will be based on solid mechanics that allow the player to customize their game experience, as well as provide a fun gaming experience. Keeping in mind how the game can become boring after replaying, it is key to develop game dynamics that deal with the issue by keeping the gameplay fresh. Complex dynamics can, therefore, solve the problem of potentially simple or boring mechanics.

Aside from the main goals of the project, the Computer Science students aimed to learn how to use Unity Engine, the game engine used during this project's development. This is more of a personal goal rather than a goal for the global project. This was the first time these students encountered video game development, so this project brought the perfect opportunity for us to finally try game development.

2.4.3 Paper Structure

The rest of the paper is divided into sections, each covering a different part of the game developing process. Section 3, called "*Preliminares*" or preface, presents the general work plan along with the tools used in this project. Section 4, "*Conceptos Básicos del Juego*" or basic game elements, explains and expands on the different mechanics and dynamics of the game, all from a user experience point of view. It also includes an example of a typical game loop to give context and describe how the mechanics of the game interact with each other in the context of a playthrough. Section 5, "*Fase de Desarrollo*" or development stage, gives a deeper understanding of the more technical aspects of the game. This is so that, combined with the previous game-design oriented explanations from the last section, a more accurate and complete vision of the game can be achieved. It touches on several different aspects of the game ranging from programming, design, art or networking. Section 6, "*Fases de Prueba*" or testing stage, presents the findings of user game testing, and how we dealt with the resulting feedback. Section 7, "*Trabajo Personal*", revolves around the team's personal experience and how it was possible to take advantage of each person's individual strengths. Lastly, sections 8 and 9. "*Conclusiones y Trabajo a Futuro*" and "*Conclusion and Future Work*" act as a closing statement for the paper as well a summary of all the previous sections, along with some suggestions for further improvements in the future.

3. Preliminares

En esta sección se tratará la planificación del proyecto. También expondremos las tecnologías involucradas en el TFG.

3.1 Planificación del desarrollo

Comenzamos dividiendo el trabajo en dos partes principales o ramas de desarrollo: por un lado el desarrollo de interfaz de usuario o IU y por otra parte el desarrollo de las mecánicas básicas del juego, con el objetivo de tener un juego funcional con las mecánicas básicas para un jugador. En estas secciones explicaremos de manera introductoria los temas ligados al desarrollo: estos temas se explicarán con mayor extensión en la sección 4 (*Conceptos Básicos del Juego*) y en la sección 5 (*Fase de Desarrollo*).

3.1.1 Planificación del desarrollo de la IU

Nos centramos tanto en los menús del juego como en la escena del propio juego, donde el jugador jugaría las canciones. Dejando el sistema de amigos para más adelante, nos centramos en las partes básicas del juego. En el transcurso de este desarrollo cambiamos múltiples veces el diseño y la estética de la IU, corrigiendo errores que tenían los diseños anteriores y haciendo mejoras visuales.

3.1.2 Prototipado de las mecánicas básicas del juego

En esta fase desarrollamos los sistemas básicos para que el juego funcionara, implementando los notas, con su sistema de aparición e interacción con el jugador. Por otro lado, hicimos la creación de los primeros MIDIs de prueba y la integración de estos al proyecto. Los MIDIs indican cuando aparecen las notas y de qué tipo son para asociarlas a una canción. Los golpes aparecen en función de un BPM (*beats per minute* o tempo) obtenido de un archivo que guarda la configuración externa.

3.1.3 Dificultades con la planificación y cambios

Al poco tiempo de empezar el proyecto tuvimos problemas con la plataforma cooperativa *Github*. Esto se debía a que las escenas de Unity son objetos muy complejos y Github no siempre hacía un buen trabajo al fusionar los trabajos de diferentes miembros del grupo, si varias personas trabajaban sobre la misma escena. Por ello, decidimos cambiarnos a la plataforma Unity Collaborate ya que al estar desarrollada específicamente para Unity no encontramos estos problemas.

Optamos por usar la biblioteca *SMFLite* dado que necesitamos leer un archivo MIDI en C#. Después tuvimos un incidente con la sincronización de las notas, y pensamos en cambiar de biblioteca a *DryWetMIDI*. En la implementación *DryWetMIDI* con el proyecto descubrimos una incompatibilidad con Android. Por lo que preferimos usar *SMFLite* arreglando el problema de sincronización.

Introdujimos la posibilidad de jugar con un mando de consola Wii. Sin embargo esto afectaba muy negativamente al rendimiento por lo que optamos por descartarla.

3.1.4 Servicio de alojamiento de archivos

Drive fue nuestro servicio de almacenamiento dado que es fácil de usar y todos lo conocíamos. Usamos este servicio para varios propósitos, entre ellos el alojamiento de archivos relacionados con el arte como pueden ser texturas o imágenes del juego. Así, cualquier miembro del grupo tendría acceso a ellas para dar su opinión o incluirlas en el juego. También almacenamos el registro de trabajo, la memoria del TFG y otros documentos, permitiendo modificaciones y visualizaciones simultáneas.

3.1.5 Plataformas de desarrollo cooperativo

Al comienzo del proyecto acordamos utilizar Github, una plataforma de desarrollo cooperativo que conocíamos todos los miembros del grupo. Por los motivos comentados anteriormente, decidimos hacer el cambio a Unity Collaborate. Unity Collaborate también es una plataforma de desarrollo cooperativo y está ideado para ser utilizado con el desarrollo de videojuegos en Unity. Esto nos permitió realizar cambios en una misma escena siempre que no editáramos simultáneamente el mismo objeto de dicha escena. También acordamos un protocolo de trabajo a la hora de repartirnos las tareas semanalmente para evitar futuros conflictos.

3.1.6 Entorno de desarrollo

Unity usa C# como lenguaje de programación, y el entorno elegido para trabajar con ello es Visual Studio. Visual Studio es un entorno de desarrollo integrado que, una vez configurado para trabajar con Unity, nos permitía depurar el código en C#. Posteriormente también lo usamos para trabajar con las bibliotecas de código *Dry Wet MIDI* y *Smflite* para implementación del sistema de lectura de los archivos MIDI.

3.1.7 MIDI

Es un estándar que define un protocolo de conexión e interfaz gráfica para ordenadores e instrumentos musicales (Huber, 2007, 1-12). Decidimos usar MIDI como contenedor para los niveles ya que nos permitía colocar notas en puntos de la canción usando notación musical. Esto es, usando pentagramas y notas musicales en vez de un formato de texto en el que las notas se representarán únicamente como eventos temporales. Usar el formato MIDI nos permitía sincronizar las notas con la música de forma más natural aprovechando la estructura de los compases.

3.1.8 Motor

Planteamos varias opciones como elección de motor. Las dos elecciones principales fueron *Unity* y *Unreal*. Elegimos *Unity* debido a su potencia y posibilidades en el desarrollo de un videojuego en 2D, además de que ciertos

miembros de nuestro grupo ya conocían esta tecnología. *Unity* también es más fácil de aprender, tiene más tutoriales y material didáctico (Unity Technologies, 2021) comparado con *Unreal*.

3.1.9 Plataforma de trabajo de audio digital

Para usar los MIDIs seleccionamos la plataforma *Reaper* (Reaper, 2021), que nos permitía crear los archivos MIDI que se asociarían posteriormente en Tamb-On a una canción. En estos MIDIs definiríamos qué tipo de notas aparecerían, en qué orden y su BPM asociado.

3.1.10 Base de datos

Nuestra base de datos es Firebase: una base de datos gratuita, compatible con Unity, sencilla, con seguimiento de errores, almacenamiento en la nube, y que ofrece autenticación de usuarios, entre otros servicios (Firebase, 2021). Sus ventajas principales respecto a otras bases de datos es que ofrece una gran cantidad de documentación, facilidad de uso, soporte gratuito y se ajusta perfectamente a lo que queremos..

3.1.11 Arte

Para el desarrollo del arte se usó principalmente el programa de código abierto Krita (KDE, 2021). Este programa es muy adecuado para el desarrollo de texturas para videojuegos ya que permite la edición de imágenes vectoriales así como la creación de texturas en formatos png u otros. Ejemplos del uso de Krita para el arte serían la creación de logos u otras formas vectoriales como pueden ser los elementos de la interfaz de usuario, o la creación de texturas de sprites mediante dibujo digital como son los personajes o los mutadores. En un momento del desarrollo también se hizo un modelo 3D de un tambor, usando para ello el programa de código abierto de edición 3D Blender. Este modelo 3D no llegó a la versión final ya que decidimos cambiar la interfaz a sprites 2D.

4. Conceptos básicos del juego

En esta sección explicaremos en qué consisten las mecánicas y dinámicas del juego, así como dar un ejemplo de un ciclo de juego o partida típica.

4.1 Mecánicas

Se puede definir mecánicas como los aspectos fundamentales del juego, lo que quedaría si se eliminan todos los elementos externos como estéticas, historias o las herramientas o tecnología del juego (Schell, 2014, 157-200). En cuanto a las mecánicas de Tamb-On, este sigue el patrón de la mayoría de juegos de ritmo: diseñar mecánicas simples de entender pero difíciles de perfeccionar. De esta

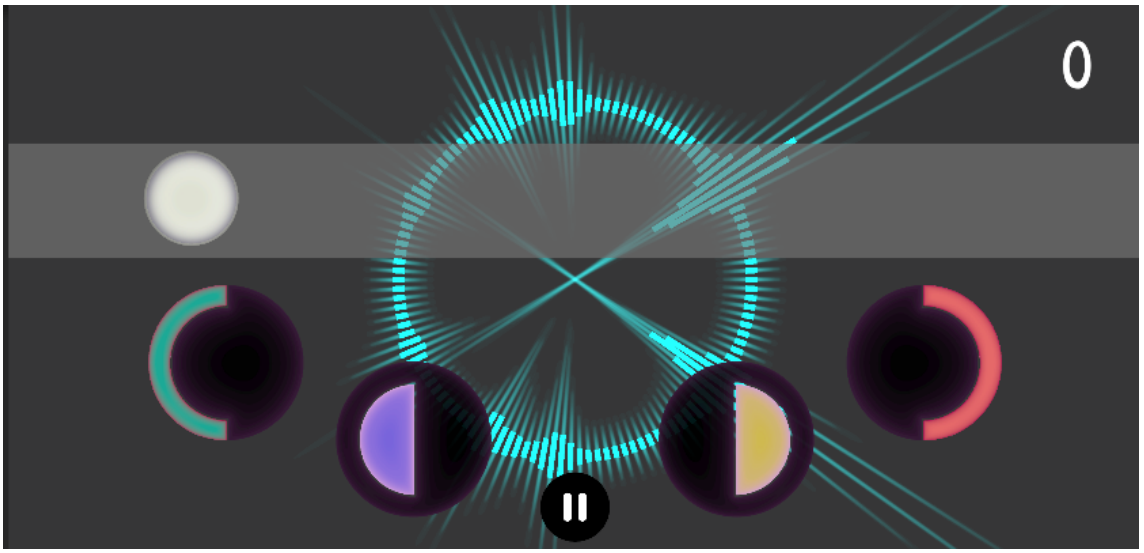
forma se alienta al jugador a seguir jugando y conseguir mejores resultados. A continuación se exponen las mecánicas de Tamb-On.

4.1.1 Golpear las notas

La mecánica central del juego consiste en golpear las notas en el momento en el que llegan al punto de golpe. Para ello el jugador dispone de unos botones en la parte inferior de la pantalla, cada botón situado de forma intuitiva según el tipo de nota que active. Además, están codificados de distintos colores y tienen la misma forma que las notas en la partida (ver figura 5).

En la versión móvil la única forma de golpear las notas es usando estos botones, pero en la versión a PC también incorporamos la opción de usar el teclado para una experiencia más acorde al uso esperado en ordenadores.

Figura 5: vista de la pantalla de juego y sus botones



4.1.2 Navegación por los menús, selección de canción

Los menús del juego dan la posibilidad de seleccionar distintas opciones para jugar tu partida. En primer lugar, seleccionamos la opción “un jugador”, ya que la opción multijugador no llegó a ser completamente desarrollada para este proyecto. Un nuevo menú se abrirá donde el jugador podrá ver distintos botones con las canciones disponibles, así como datos sobre las mismas (ver figura 6). Resaltado en letra blanca en la parte superior del botón se encuentra el nombre de la canción. También se puede ver una puntuación en base a 5 estrellas que usamos para describir la dificultad. A la derecha de la dificultad se puede ver el autor de la canción así como la duración en minutos y segundos de la canción. Una vez pulsado uno de los botones de selección de canción, se procederá a la siguiente pantalla. En todo momento hay también disponibles botones para retroceder o salir del juego, consistentemente situados en la parte superior izquierda.

Figura 6: vista de la pantalla de selección de canción

4.1.3 Registro de usuarios

Para acceder al sistema de amigos o a los récords, el jugador tiene primero que registrarse en nuestro sistema. Registrarse es sencillo: solo hay que clicar en el botón "iniciar sesión" en la pantalla de inicio del juego. Este botón te lleva a un menú donde clicando sobre otro botón "registrarse" podrás crear una cuenta de usuario proporcionando una dirección de correo electrónico, un nombre de usuario y una contraseña (ver figura 7). Si ya tienes una cuenta, basta con introducir tu correo y contraseña en sus campos correspondientes y pulsar el botón verde de "iniciar sesión".

Figura 7: vista de la pantalla de registro de usuario

4.1.4 Visualización de récords

Cuando el jugador completa una partida y tiene la sesión iniciada las puntuaciones que consiga se guardan en un registro de récords (ver figura 8). Cualquier usuario puede visualizar estos récords canción por canción, pero si además está registrado podrá saber sí ocupa un lugar entre las mejores puntuaciones. La mejor puntuación del jugador se mostrará en la parte inferior de la pantalla, además de datos adicionales como qué puesto ocupa en el ránking.

Figura 8: vista de la pantalla de récords

4.1.5 Sistema de amigos

Los jugadores registrados pueden acceder al sistema de amigos, gracias al cual podrán ver quién está en línea o jugando una partida en cada momento. Para acceder al menú de amigos solo hay que pulsar el botón “amigos” situado en la parte inferior del menú principal (ver figura 9). Al pulsarlo se desplegará un menú en el lateral derecho en el que aparece una lista de los amigos añadidos, así como un espacio para notificaciones de amistad y un campo para añadir amigos buscándolos por su nombre de usuario. Para cerrar el menú desplegable, el usuario debe pulsar otra vez el botón “amigos”, que ahora habrá cambiado su descripción a “cerrar amigos”.

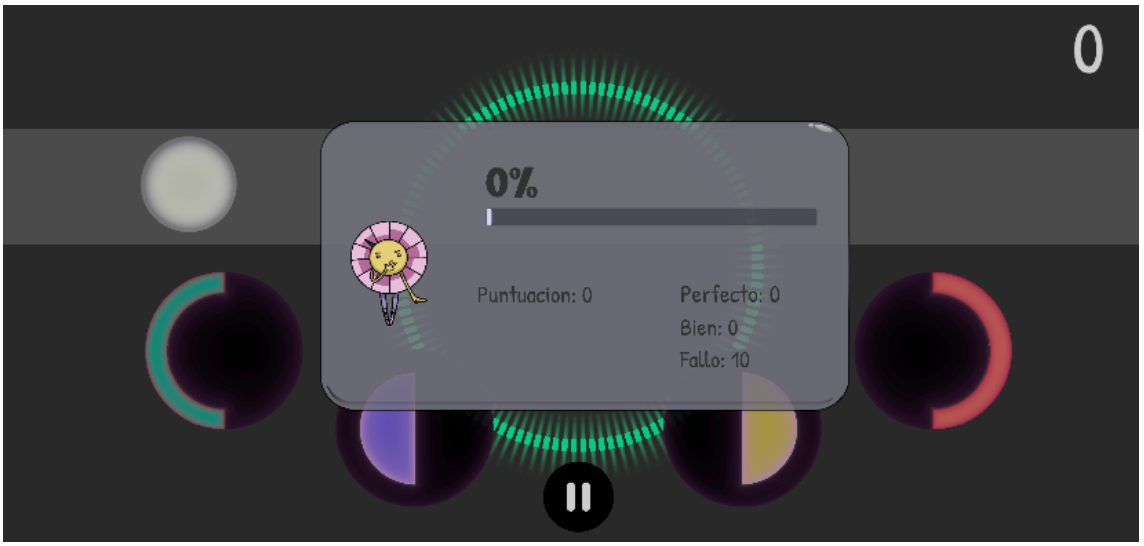
Figura 9: vista de la pantalla de amigos

4.1.6 Menús de pausa y final de partida

Durante la partida, hay dos posibles menús que pueden aparecer. Ambos tienen visualmente muchas similitudes pero cumplen funciones distintas. En primer lugar, el menú de pausa aparece cuando el jugador pulsa el botón destinado a ello, situado en la parte inferior central de la pantalla (ver figura 10). En él se dan tres opciones: continuar la partida, reiniciarla o salir al menú principal. El menú de fin de partida, por otro lado, aparece automáticamente cuando la canción acaba y en ella el jugador puede ver datos sobre su rendimiento como pueden ser la cantidad de golpes perfectos acertados, o el número total de puntos, entre otros (ver figura 11). Tocar sobre cualquier parte de este menú nos llevará a la pantalla de selección de canción, en el menú principal.

Figura 10: vista de la pantalla de pausa

Figura 11: vista de la pantalla de fin de partida

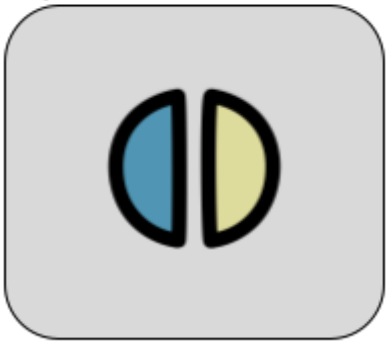


4.1.7 Mutadores

Procedemos a describir los distintos tipos de mutadores que podemos incluir. Es obligatorio elegir uno de cada par, pero si el jugador no selecciona ninguno, ciertos mutadores estarán seleccionados por defecto dependiendo del personaje escogido.

Figura 12: mutador "difícil"

Figura 13: mutador "fácil"



Difícil (figura 12): En el modo difícil hay 4 tipos de notas: derecha interior y exterior e izquierda interior y exterior. Supone un desafío extra para los jugadores más experimentados. En la versión PC, se usan las mismas teclas que en el modo fácil para las notas interiores, y Q y P para las exteriores izquierda y derecha respectivamente

Fácil (figura 13): En el modo fácil solo hay dos tipos de notas: derecha e izquierda. Es ideal para practicar o para quien prefiere una experiencia de juego más relajada. En la versión PC, se usan las teclas F para la izquierda y J para la derecha

Figura 14: mutador "combo"

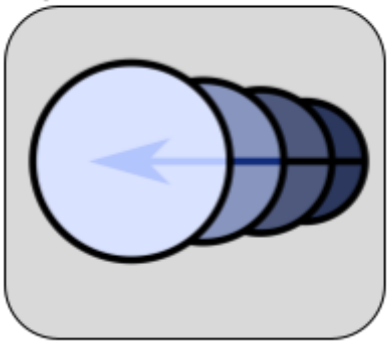


Figura 15: mutador "precisión"

Combo (figura 14): El modo combo premia los aciertos consecutivos. Cuantos más encadenes seguidos, mayor bonus de puntos consigues. Sin embargo, si fallas una nota el combo se reinicia y pierdes el bonus.

Precisión (figura 15): El modo precisión valora más las notas bien dadas individualmente sin tener en cuenta el combo: todas las notas valen más, pero su valor es constante durante toda la partida.

Figura 16: mutador "inmortal"

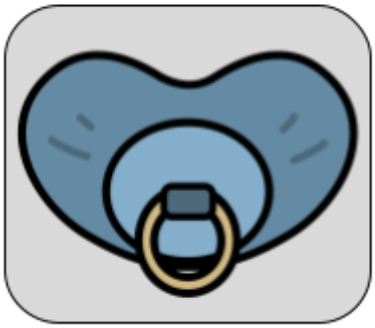


Figura 17: mutador "mortal"

Inmortal (figura 16): El modo inmortal es ideal para practicar las canciones y familiarizarte con ellas, ya que no pierdes la partida aunque falles mucho. Sin embargo, hay una gran penalización en la puntuación.

Mortal (figura 17): El modo mortal es el estándar. Te da más puntos pero también significa que perderás la partida si fallas un cierto número de golpes seguidos.

Figura 18: mutador "ciego"

Figura 19: mutador "visible"

Ciego (figura 18): El modo ciego añade un componente extra de dificultad: las notas solo son visibles cuando están cerca del punto de golpe. Proporciona bonus de puntos, pero es un desafío.

Visible (figura 19): El modo visible es el estándar. Las notas aparecen desde el final de la zona de juego, por la derecha, y se puede ver cómo se acercan a la zona de golpe. No proporciona puntos extra.

4.1.8 Personajes

Por otro lado también es obligatorio escoger un único personaje de los cinco que damos a elegir. Cada personaje tiene un conjunto de habilidades y características que otorgan al jugador ciertos beneficios pero también sus propios inconvenientes.

Figura 21: "Floratoria"

Figura 20: "Caxper"

Floratoria (figura 21): al seleccionar este personaje, una nota aleatoria aparecerá cuando comiences la partida. Conseguirás puntos extra si aciertas todas las notas del tipo que ha aparecido.

Caxper (figura 20): solo se puede jugar en modo combo por lo que el mutador precisión no se podrá seleccionar. Además, fallar notas seguidas penaliza en la puntuación, pero hacer combo puntúa extra.

Figura 23: "Mister Papa"

Figura 22: "Silvestre"

Mister Papa (figura 23): seleccionar este personaje hará que recibas mutadores aleatorios para esa partida, cada mutador activado proporciona una bonificación ligera sumada a su bonificación principal.

Silvestre (figura 22): si juegas con este personaje solo se contarán los golpes perfectos para la puntuación. Es el personaje más exigente. Cada perfecto puntúa más en comparación con otros personajes.

Figura 24: “Topita”

Topita (figura 24): todos los golpes valen como un bien, tanto los golpes “bien” como los perfectos. Ganarás menos puntos.

4.2 Dinámicas y ciclo de juego

En esta sección se explica cómo las interacciones entre distintas mecánicas conforman las dinámicas, y a su vez las dinámicas en el contexto de la aplicación crean la experiencia de juego.

4.2.1 Dinámicas

Las dinámicas principales del juego aparecen durante la propia canción a jugar. Consisten en las interacciones de los personajes y mutadores además del rendimiento del jugador, para conseguir una puntuación final. Así, ciertos personajes pueden favorecer que el jugador preste más atención a una nota en particular (como puede ser Floratoria) sin preocuparse tanto por el combo, mientras que otros pueden poner más énfasis en mantener un combo (como Caxper). Estas interacciones dan forma a diferentes estilos de juego y diferentes enfoques, con lo que la dinámica del juego es muy variada y personalizable. Mismas mecánicas como pueden ser golpear una nota pueden tomar matices distintos si comparamos por ejemplo una partida usando “Silvestre” y “Mortal” con una usando “Topita” e “Inmortal”. La mecánica de golpear la nota es exactamente la misma, pero las interacciones de los mutadores y personajes crean una dinámica tensa y desafiante para el primer ejemplo, y una dinámica relajada y divertida para el segundo.

4.2.2 Ciclo de juego o partida típica

El juego comienza con el jugador abriendo la aplicación, tras lo cual aparece el menú principal con diferentes opciones e interacciones. Para jugar una partida basta con tocar el botón “un jugador” en este menú, con lo que se nos desplegarán diferentes opciones para configurar nuestra partida. El jugador seleccionará una de las canciones a elegir, dependiendo de sus preferencias de dificultad o duración, y pasará a la selección de personajes y mutadores. Estas selecciones se hacen de

forma lineal por lo que se promueve el pensamiento estratégico del jugador al presentársele cada partida con las posibilidades de los diferentes personajes y mutadores. Una vez los haya elegido, la canción comenzará. Se pasará a otra pantalla en la que se pueden ver los botones de control en la parte inferior de la pantalla, y en la que las notas a golpear aparecen en una línea. Cuando las notas se superponen con el objetivo blanco situado a la izquierda de esta línea, el jugador ha de pulsar el botón que corresponda a la nota para así conseguir puntos. La canción transcurrirá de esta manera, mientras el jugador consigue puntos dependiendo de las opciones de personalización seleccionadas anteriormente. Al llegar el final de la canción, el menú de final de partida aparecerá automáticamente haciendo saber al jugador cómo de bien le ha ido. Tras consultar su puntuación, el jugador tocará en la pantalla y el juego volverá a la selección de canciones para facilitar la rejugabilidad. Si el jugador quisiera consultar los récords, podría usar la flecha de "volver" para regresar al menú principal y tocar sobre la opción "récords". Cuando desee dejar de jugar, solo habría que pulsar el botón de "volver" para ir regresando a los menús anteriores hasta que en el menú principal el botón se muestra como una cruz. Al pulsar esta cruz, el juego se detendrá y se cerrará la aplicación.

5. Fase de desarrollo

Para el desarrollo del juego usamos una metodología circular similar a la SCRUM (Pries & Quigley, 2010, 40-79), en la que cada semana nos reuníamos para discutir qué elementos serían desarrollados esa semana. Gracias a este sistema circular tuvimos un prototipo semi-funcional en una etapa temprana del desarrollo. Muchas de las secciones que se exponen a continuación se desarrollaron en paralelo, aprovechando además las fortalezas de nuestro equipo multidisciplinar.

5.1 Diseño del sistema de puntuación

El sistema de puntuación depende de 3 factores: El personaje seleccionado, los mutadores activos y la precisión con la que se dé a la nota. Se obtiene puntuación cada vez que se pulsa correctamente el botón correspondiente a una nota en el tiempo adecuado, es decir, cuando llega al círculo blanco. Distinguimos 2 casos:

- En el momento del golpe la nota está 100% incluida en el círculo blanco se considera "perfecto" y tiene un valor base de 30 puntos.
- En el momento del golpe la nota está más de un 0% y menos de un 100% incluida en el círculo blanco se considera "bien" y tiene un valor base de 10 puntos.

Estos valores son modificados por mutadores y personajes.

5.1.1 Mutadores

En cada mutador hay 2 opciones, donde la opción por defecto no modifica el valor y la restante le aplica un multiplicador acumulativo, positivo o negativo dependiendo del tipo de efecto:

- **Fácil/Difícil:** Fácil es el modo por defecto. Difícil, al suponer una desventaja, hace que el valor base obtenido se multiplique por 1.2.
- **Visible/Ciego:** Visible es el modo por defecto. Ciego, al suponer una desventaja, hace que el valor base obtenido se multiplique por 1.3.
- **Mortal/Inmortal:** Mortal es el modo por defecto. Inmortal, al suponer una ventaja, hace que el valor base obtenido se multiplique por 0.5.

Todos estos valores se acumulan para obtener el modificador final. De modo que, por ejemplo jugando con la mayor desventaja posible ("difícil", "ciego", "mortal") se puede obtener la máxima puntuación (1.2*1.3*1 = 1.56). Obsérvese que utilizando todas las desventajas, si se selecciona el modo "inmortal" el valor se multiplicaría por un valor inferior a 1 (1.2*1.3*0.5=0.78). Esto es debido a que consideramos que "inmortal" es un modo de práctica y no debe dar una oportunidad clara para entrar en los récords mundiales. En este lugar deben figurar las partidas que presenten un gran desafío.

Existe otro mutador, el "combo"/"precisión". Sin embargo, este es manejado de forma distinta. El combo depende de un valor que cambia a lo largo de toda la partida; es modificado cada vez que se acierta o falla una nota y no es un valor fijo en cada iteración. Se suma 1 (o 2 si es un perfecto) a un valor acumulado según cuantas notas acertadas seguidas se lleven más el valor base que corresponda, es decir, supongamos que no hay activo ningún mutador además de éste, si se llevan 4 notas "bien" seguidas y se acierta una quinta valorada con "perfecto" la puntuación sumaría 30+5=35.El valor acumulado por combo se resetea a 0 si se falla una nota. "Precisión" es el contrapunto a este beneficio, al no poder obtener este valor acumulado el valor base para los perfectos pasa de 30 a 50. 5.1.2 Personajes

Los personajes interactúan con los mutadores provocando cambios en las reglas previamente descritas.

- **Topita**: Al no distinguir entre perfectos y bien existe un único valor base "20" el cual sigue siendo susceptible a los modificadores.
- **Silvestre**: No se obtienen puntos por "bien", el valor base de "perfecto" pasa a ser 60 con combo activado y 100 con precisión activado.
- **Caxper**: El valor acumulado con combo activado (obligatorio para usar este personaje) pasan de ser 1 y 2 en cada iteración a ser 2 y 4 respectivamente.
- **Mr. Papa**: Suma 0.1 al multiplicador final por cada mutador que haya activado aleatoriamente.
- **Floratoria**: La puntuación en caso de cumplir la condición del personaje viene dada por la siguiente fórmula:

Puntuación final = Puntuación actual + ((Puntuación actual / 100) * 30)

5.2 Diseño de MIDIs definitivos e integración

Se recibe por fichero MIDIs una serie de instrucciones midi que son traducidas por la biblioteca *SmfLite*. Donde podemos acceder a información clave sobre cada nota. Podemos saber mediante valores hexadecimales el tipo de nota que se recibe y su escala. Para poder crear estos ficheros MIDI con un programa externo adjuntamos como anexo una guía sobre este tema. La notación que usamos para hacer la traducción al nuestros objetos del juego son:

- Do3: Botón de abajo a la izquierda.
- Do#3: Botón de abajo a la derecha.
- Re3: Botón de arriba a la izquierda.
- Re#3: Botón de arriba a la derecha.

Sin embargo, averiguar el momento en que se debe instanciar el objeto de cada nota para que sea síncrono con el audio es más complejo.

Las instrucciones MIDI tienen un tiempo relativo entre ellas, pero no un tiempo real. Si tenemos acceso al BPM ("beats per minute" o tempo) al que se reproduce el audio de la canción original sí se puede obtener el tiempo real en el que debería aparecer una nota.

Finalmente se añaden a una cola todos los eventos MIDI procesados con el tipo de botón y el tiempo exacto en el que se debe instanciar. Este tiempo equivale al tiempo real en el que la nota es tocada en la canción, más el tiempo que se necesita para que el objeto recorra la pantalla de derecha a izquierda hasta el centro del círculo blanco, resultando así que la nota llegará exactamente a donde debería ser pulsada en el momento exacto en el que se oye en la canción.

5.3 Diseño del sistema de amigos

Cada usuario tiene tanto una lista de amigos como una lista de solicitudes de amistad. Las solicitudes se pueden aceptar, añadiendo así a la persona a la lista de amigos, o rechazarla. En ambas opciones se elimina al sujeto de la lista de solicitudes.

Se puede mandar una solicitud de amistad a alguien con un nombre existente en la base de datos con las excepciones de no poder añadirse a uno mismo, a una persona que ya exista en su lista de amigos, o una persona que tenga una solicitud de amistad pendiente suya.

Además en la pestaña de amigos se mostrará una lista de todos los amigos del usuario junto con un estado: "disponible" si está activo en la aplicación, "ocupado" si está jugando una canción, y "desconectado" si no tiene la aplicación activa.

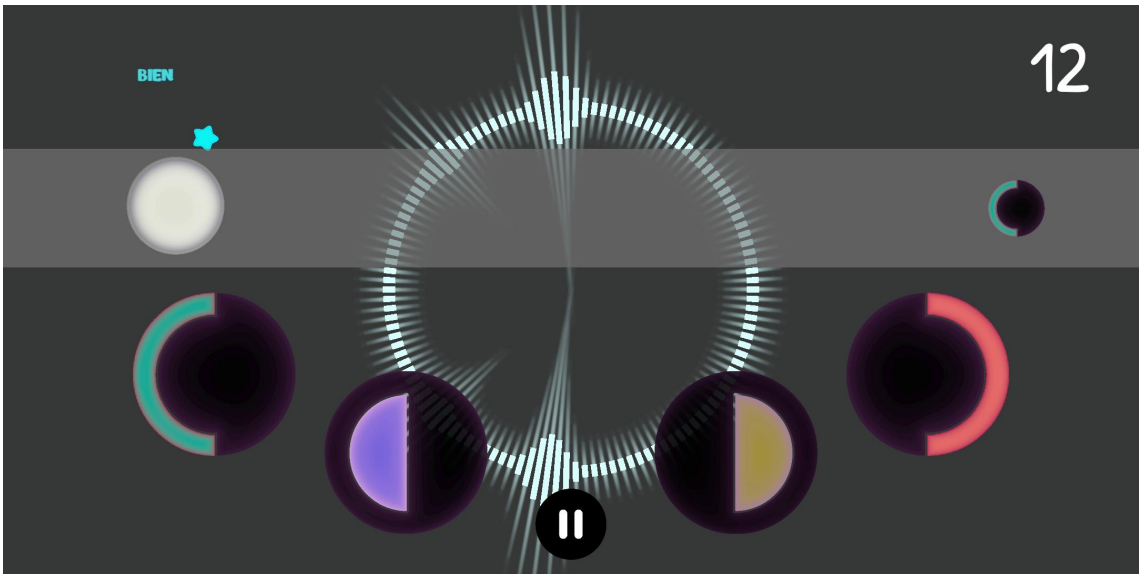
5.4 Diseño de la tabla de puntuaciones

Al acceder a la pantalla de récords se visualizan las 10 puntuaciones más altas junto con sus datos (mutadores activos, personaje utilizado, nombre de usuario) además de una última fila en la que se muestra el récord personal en caso de haber jugado dicha canción junto con la posición que ocupa en los récords absolutos. Aunque solo se muestran los 10 primeros récords absolutos, en la base de datos se almacenan hasta 100 por lo que un usuario puede observar que esté en la posición 56, por ejemplo, pese a que no aparezca en los récords mundiales. Si tiene una marca personal pero no incluida dentro de los récords absolutos saldrá como "+100". Dentro de esta pantalla se puede navegar entre cada canción donde se ven sus datos con este criterio.

5.5 Efectos especiales

Los efectos especiales y animaciones proporcionan un feedback visual claro al usuario. Además, ayudan a dar dinamismo al juego y a crear cierto sentido de diversión. Podemos dividirlos en dos categorías: efectos de partículas y animaciones (ver figura 25).

Figura 25: pantalla de juego en la que se pueden ver varias animaciones y efectos



5.5.1 Efectos de partículas

El juego cuenta con varios efectos de partículas de Unity para producir efectos especiales. En primer lugar y quizá el más visible, el efecto de líneas de luz del menú principal está creado con un sistema de partículas en el que cada una deja una cola de luz a su paso. El color de la luz está modulado usando un gradiente con los colores del arcoíris. Este efecto tiene un número limitado de partículas y está

diseñado para no resultar una gran carga al sistema, ya que está emitiendo constantemente.

Otro efecto de partículas es la "explosión de estrellas" que aparece cuando el jugador golpea una nota correctamente. Es un efecto muy rápido pero visualmente agradable, y proporciona feedback al jugador de si ha golpeado una nota en su sitio. El efecto está compuesto de un sistema de partículas que crea dos o tres estrellas en un tiempo reducido, y las manda hacia arriba con un ligero ángulo. Además, les aplica el mismo efecto de gradiente arcoíris para añadir variedad. La textura de las estrellas es de creación propia.

Para el efecto de barras dinámicas que se puede ver en el fondo mientras se juega una canción, decidimos importar un asset de la tienda de Unity: Rhythm Visualizator Pro. Importar un *asset* era más eficiente en este caso ya que crear algo similar desde cero habría requerido un esfuerzo extra y dado que teníamos tiempo limitado, preferimos dedicar nuestro tiempo al desarrollo del juego en sí. El efecto está configurado por nosotros, y reacciona dinámicamente a la canción, haciendo que ciertas barras aumenten de tamaño y que todo el sistema vaya cambiando de color siguiendo un gradiente con varios colores.

5.5.2 Animaciones

Los botones dentro del menú realizan una animación cuando son pulsados, reproducen un sonido corto y el tamaño se expande y contrae en un pequeño espacio de tiempo.

Aparecen animaciones sobre un texto cuando hay un evento de "fallo", "bien" o "perfecto" durante la partida. Están resaltados en los colores rojo, azul y verde, respectivamente. Se colocan encima del círculo blanco y se desplazan hacia arriba con un ligero efecto de rebote, acto seguido se desvanecen.

Existe otra animación muy similar cuando está activado el mutador de combo. El texto indica el número de notas seguidas actual y dicho texto se expande y se contrae brevemente para quedarse estático durante unos segundos hasta desaparecer.

Si se juega con el personaje de Floratoria al comienzo de una partida se le indica mediante una animación cuál es la nota deseada por el personaje. La animación consiste en 2 imágenes de la nota que se desplazan desde la izquierda y la derecha hacia el centro de la pantalla para fusionarse, expandirse y finalmente desplazarse hacia arriba hasta fuera de los límites de la pantalla.

5.6 Diseño y arte de mutadores y personajes

El diseño de los elementos del juego se hizo de forma incremental, siguiendo un proceso de lluvia de ideas, finalización de la idea, abocetado y creación de la textura para su integración el juego. La producción del arte se llevó a cabo tanto de

forma tradicional en la fase de abocetado como en digital usando Krita en la fase de finalización y creación de la imagen para el juego.

5.6.1 Diseño y producción de mutadores

Los mutadores se diseñaron con la idea de crear una imagen final simple y reconocible usando formas sencillas. Los significados de los mutadores también se reflejan en el diseño, utilizando conceptos fácilmente reconocibles para ello. Para su producción se decidió usar arte vectorial, ya que la mayoría de los mutadores tienen formas relativamente geométricas.

Para los mutadores de "fácil" y "difícil" se usa una representación visual simplificada de las notas existentes en el juego. Consisten en un objeto de forma redondeada, dividida en 4 trozos en el caso del mutador de modo difícil y en 2 en el de modo fácil. Esto coincide con el resultado de elegir cada mutador, por lo que resulta fácil e intuitivo para el jugador entender el significado de cada mutador.

Para el mutador de **combo** se utiliza una serie de elementos circulares en sucesión superpuestos con una flecha. Estos círculos representan las notas, y se usa la idea de "encadenarlas" para conseguir mayor puntuación. El mutador de **precisión** es su contrario, y está diseñado para parecer una diana. Se eligió una diana ya que es un objeto que se asocia fácilmente a la precisión o dar en el blanco.

El mutador de **mortal** usa un simbolismo muy conocido, el de las calaveras. Su opuesto, el modo **inmortal**, está inspirado en un chupete de bebé. El contraste con la calavera es lo que le da mayor significado, al tener lado a lado algo inofensivo y algo peligroso.

Para el modo **ciego** decidimos usar otro símbolo usado con frecuencia en los videojuegos: unas gafas de sol. Tuvimos una conversación acerca del nombre del mutador, ya que es potencialmente ofensivo para las personas con visibilidad reducida. Se decidió usar ciego porque es la traducción directa de "blind", usado ampliamente en videojuegos para efectos parecidos al nuestro en Tamb-On. El modo opuesto es el **visible**, representado de forma obvia para el jugador con un dibujo de un ojo abierto.

5.6.2 Diseño y producción de personajes

Al diseñar los personajes se tuvo en cuenta la "personalidad" de cada uno, dependiendo de los efectos que proporcionaran. Tanto el aspecto visual como el nombre del personaje se diseñaron para que de alguna manera estuvieran unidos al efecto del mismo. Las ideas se presentaron en una reunión de grupo, y tras decidir por un diseño abocetado se empezó su producción.

El personaje de **Floratoria** al basarse en el azar está inspirado en una ruleta, por lo que el tutú/pétalos están diseñados con divisiones además de incluir un pequeño puntero. Tiene una estética rosa y lleva zapatos de ballet. Su nombre también juega con esta idea: "flor" + "aleatoria".

Caxper está basado en un fantasma clásico con la bola y la cadena. La idea para su diseño vino del hecho de que los fallos seguidos penalizan a la puntuación, de ahí que en la bola del personaje como metáfora se lea de forma sutil "fail", fallo en inglés. El nombre es un guiño al conocido personaje de las películas.

Mister Papa es un personaje que selecciona los mutadores de forma aleatoria para cada partida. Basado en esto, el diseño hace referencia a un popular juguete conocido por poder intercambiar sus accesorios y darle diferentes apariencias constantemente. El arte incluye referencias a varios mutadores existentes en el juego, como guiño a que no sabes qué mutador te tocará en esta partida además de recordar esa idea de "probarlo todo" tan presente en la inspiración para el diseño.

Silvestre es un personaje que podría considerarse desafiante o difícil. En la reunión de diseño, tras una lluvia de ideas, se acordó crear un personaje que visualmente fuera adorable pero con un toque peligroso. Silvestre es un gatito con una apariencia muy dulce pero con un cuchillo escondido tras la espalda, haciendo referencia a la dificultad del personaje en el juego. El nombre hace referencia a un conocido de dibujos animados.

Por último, **Topita** es un personaje en el que la precisión no importa tanto ya que todos los golpes (ya sean "bien" o "perfecto") valen como si fueran lo mismo. El diseño está inspirado en la visión pobre de los topos, con la idea de que este personaje es perfecto para aquellos con "vista de topo" o mala precisión. En el dibujo, Topita tiene unas gafas redondas como centro de atención del personaje. Como las gafas son el punto central, el nombre está basado en un conocido personaje de animación japonesa que lleva gafas, mezclado con la palabra "topo".

5.7 Scripts a destacar

Durante el desarrollo, seguimos unos estándares de código y estilo para asegurar homogeneidad en el mismo. En primer lugar, decidimos programar usando el inglés para los nombres de los métodos ya que en el desarrollo de videojuegos el inglés es el estándar. Además, seguimos las guías de estilo de C# (Olsson, 2013) a la hora de nombrar objetos en el código: *CamelCase* para métodos y clases y *camelCase* para nombres de variables. Nuestras clases están organizadas siguiendo una plantilla de estilo como se muestra en la figura 26 y en el anexo 2. Se usan regiones de C# para separar distintas zonas del código, en especial una región Variables y una región *Methods*. Dentro de la región *Variables* hay otras tres sub

Figura 26: nuestra plantilla de estilo para las clases C#

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


// COPY THE INSIDE OF THIS CLASS WHENEVER YOU'RE CREATING A NEW CLASS, TO
// KEEP THE STYLE CONSISTENT ACROSS THE PROJECT AND ALLOW FOR EASY SCALLING
public class StyleTemplate : MonoBehaviour
{
    #region ---- VARIABLES ----

    #region --- PRIVATE ---

    #endregion
    #region --- PROTECTED ---

    #endregion
    #region --- PUBLIC ---
    #endregion

    #endregion


    #region ---- METHODS ----
    #region --- UNITY METHODS ---
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
```

regiones, una para cada nivel de accesibilidad de las posibles variables de la clase. Así se facilita el correcto escalado de las clases, evitando una mezcla confusa de variables con distintos niveles de accesibilidad. En la región de Métodos hay dos subsecciones: una para los métodos de *Unity* y otra para los métodos creados especialmente para las clases de Tamb-On.

5.7.1 Managers

El código utiliza un sistema de *managers* para llevar el control de los puntos más importantes de la lógica del juego. De esta manera centralizamos las funcionalidades principales y proporcionamos *scripts* capaces de comunicar la información de forma eficaz. Destacamos:

- *RhythmManager.cs*: Realiza la lectura del archivo MIDI y transforma las instrucciones MIDI en objetos dirigidos a una cola del tipo *Spawner*.
- *Spawner.cs*: Instancia un objeto *BaseButtonBehaviour* en función de la cola obtenida previamente y en el momento del debido de tiempo de la canción.
- *ScoreManager.cs*: Controla todas las operaciones matemáticas para ir obteniendo a tiempo real la puntuación de un jugador.
- *InputManager.cs*: Se encarga de recibir la entrada del usuario, ya sea de teclado en PC como botón táctil en móvil. Es capaz de determinar si una nota ha sido "perfecto", "bien" o "fallo" además de controlar las animaciones resultantes.
- *GameManager.cs*: Controla y hace de nexo entre los anteriores *managers*. Posee además información necesaria tanto para la escena de la partida como para la del menú. Es un *script* persistente durante toda la ejecución del juego.

5.7.2 Menu

Los scripts dentro de la carpeta de menú contienen operaciones para el correcto flujo desde que se inicia la aplicación hasta jugar realmente una canción, incluye comunicación entre el apartado de mutadores con el de personajes.

5.7.3 Records

El script *RecordsOperations* maneja el movimiento dentro de la pantalla de records junto con una lectura en JSON desde la base de datos, traducida con *RecordsDB*.

5.7.4 Control

- *AuthControl*: Lleva la inicialización de las dependencias de Firebase, como por ejemplo, guardar datos en caché. Gracias a esta clase no es necesario iniciar sesión cada vez que se inicia la aplicación.
- *Control_Friends*, *Control_Login_Control_Register* y *Control_StartMenu*: Controlan las operaciones a nivel de base de datos de las operaciones

relacionadas con la lista de amigos, el inicio de sesión, el registro, y control de conexión/desconexión, respectivamente.

5.7.5 Buttons

Cada script dentro de la carpeta *Buttons/InGame* tiene el comportamiento peculiar de cada tipo de botón heredando de una clase llamada *BaseButtonBehaviour* sus operaciones comunes.

5.7.6 SmfLite

Traduce el fichero MIDI a una clase que podemos manejar desde *RhytymManager.cs*.

5.8 Base de datos

Usamos como base de datos y método de autenticación la plataforma *Firebase*, debido a la amplia integración que posee con *Unity*, además de presentar más funcionalidades útiles que se podrán usar en el futuro. Como, por ejemplo, una nube con suficiente capacidad para albergar un gran número de canciones.

La base de datos la nombramos "TaikoUCM" y la podemos dividir en 3 módulos de datos. "Songs", "users" y "userSongs" que presentan dependencias entre ellos y se comunican de forma segura gracias al sistema de autenticación.

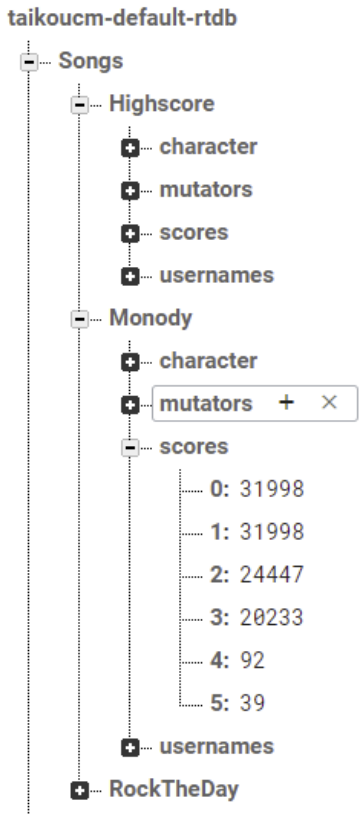
A continuación los describimos en detalle además de adjuntar una imagen descriptiva de cada uno.

5.8.1 Módulo Songs

Contiene un listado de canciones la cual incluye las puntuaciones más altas de las respectivas canciones junto con otros datos útiles. En caso de añadir una nueva canción el código automáticamente añade una nueva colección con el nombre de dicha canción y los campos necesarios, inicialmente vacíos. La estructura es la siguiente (ver figura 27):

- **songname**
  - **character**: contiene una lista (máximo 100) de los personajes usados en los récords establecidos.
  - **mutators**: contiene una lista (máximo 100) de la lista de mutadores usados en récords establecidos. Para evitar hacer un nivel más la lista de mutadores está codificada con un simple número entero, descodificado a nivel de código.
  - **scores**: contiene una lista (máximo 100) de las puntuaciones en los récords establecidos.
  - **usernames**: contiene una lista (máximo 100) de los nombres de usuarios en los récords establecidos.

Figura 27: Desglose del módulo records



Cada récord individual (con *character*, *mutators*, *scores*, *usernames*) está ordenado en la lista en función de "scores" de forma descendente. En caso de sobrepasar el máximo, se añade el récord en la posición que le corresponda, por ejemplo 47 y se elimina el que esté en la posición 100.

En caso de introducir una nueva canción se añade automáticamente a la lista con su "songname" correspondiente.

Debido a que ofrecemos la posibilidad de jugar sin registro, para poder ingresar en estas tablas el usuario debe estar registrado y haber iniciado sesión. Esto es necesario para poder escribir un nombre de usuario válido en *username*. Obtenido a través de la colección "users".

Se hace una lectura completa del módulo al acceder a la pantalla de "Records", accesible para cualquier usuario esté registrado o no.

5.8.2 Módulo Users

Contiene una lista de usuarios con información útil en cada uno de ellos (ver figura 28). Su estructura es:

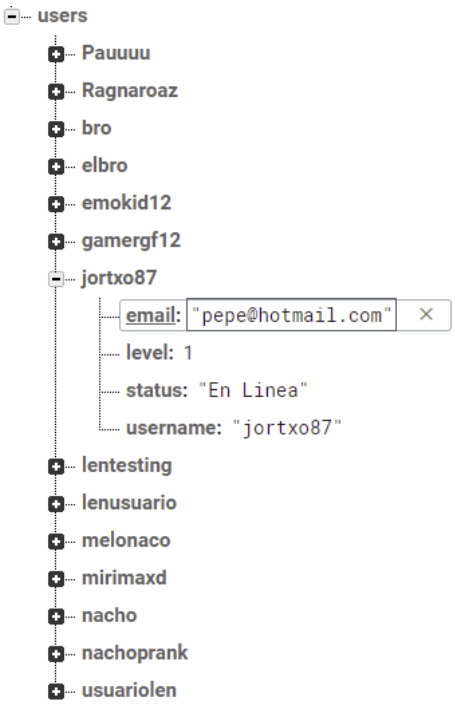
- **nombre:**
  - **email**: correo electrónico del jugador.

- **level**: nivel actual.
- **status**: Varía entre “En línea”, “Ocupado” y “Desconectado”. “En línea” en caso de estar activo en la aplicación, “Ocupado” en caso de estar jugando una canción y por último “Desconectado” si la aplicación está inactiva.
- **username**: Nombre de usuario del jugador.
- **friendRequest**: una lista con los nombres de usuario que hayan solicitado amistad al usuario actual.
- **friends**: lista con los nombres de usuario a los cuales se ha aceptado una solicitud de amistad.

“username” es el campo que se utiliza más tarde para establecer los récords.

Este módulo es usado en las ventanas tanto de registro como de inicio de sesión. En registro se escribe una nueva entrada de la colección y al inicio de sesión una lectura. Vienen condicionadas por la autenticación explicada más adelante.

Figura 28: Usuarios en el módulo Users



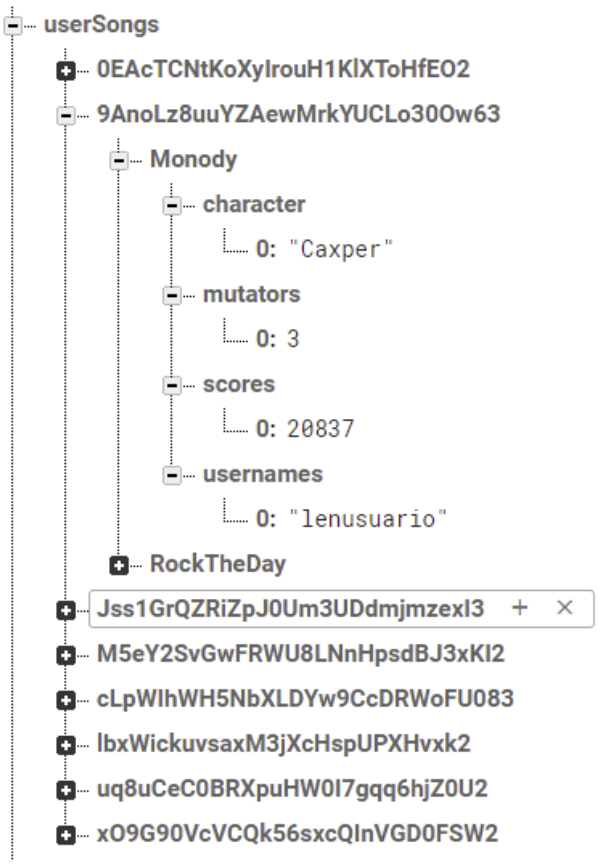
5.8.3 Módulo UserSongs

Es similar al módulo de Songs. Sin embargo, esta contiene una capa superior con el ID de cada usuario y una colección “récord” en cada uno de los usuario por cada canción que haya jugado. Contienen sus récords personales (ver figura 29). La estructura es como sigue:

- **id:** Identificador de usuario
  - **songname:** Nombre de la canción

- **character**: Personaje utilizado cuando se estableció la puntuación más alta.
- **mutators**: Lista de mutadores (codificada) cuando se estableció la puntuación más alta.
- **scores**: La puntuación más alta que se obtuvo.
- **username**: Nombre de usuario correspondiente al identificador.

Figura 29: Desglose de las canciones jugadas por un usuario



Cada ID es accedido únicamente por su propietario y solo puede ser modificado por este.

Se realiza una lectura cuando se accede a la ventana de récords en la que se puede ver un subapartado con el récord personal de cada usuario junto con la posición que ocupa en el ranking general (+100 si no está entre los 100 primeros).

5.8.4 Módulo autenticación

Permite la comunicación segura entre la base de datos y cada usuario. Se puede agregar un usuario nuevo a través de la pantalla de registro en la que se deben proporcionar la siguiente información:

- **Un proveedor:** En nuestro caso usamos el método del correo electrónico, sin embargo también es posible usar un número de teléfono o aplicaciones externas como por ejemplo *Twitter*. Las condiciones para ser aceptado son:
    - El correo electrónico no debe estar registrado.
    - Debe ser un correo válido, es decir, del formato *****@***.**
- **Un nombre de usuario:** El nombre con el que desee identificarse el usuario. Las condiciones para ser aceptado son:
    - El nombre del usuario no debe existir en la base de datos.
    - No puede contener caracteres especiales como "/" o símbolos de un alfabeto distinto al latín occidental.
- **Una contraseña:** Es una contraseña censurada y encriptada a la cual solo tiene acceso aquel con el correo electrónico al que se le asigne, nadie más puede acceder a ella. La condición para ser aceptada es:
    - Debe tener más de 8 caracteres.

La información adicional que proporciona firebase (ver figura 30) son:

- La fecha de creación.
- La fecha del último acceso.
- UID (identificador de usuario) único de cada usuario. Utilizado tanto en el módulo "UserSongs" como en el de Users para garantizar una conexión segura y garantizar la protección de los datos personales. Ciertos campos de estos módulos no son accesibles si el UID no es el adecuado.

Ofrecemos características adicionales como la recuperación de contraseña, incluida en la pantalla de inicio de sesión.

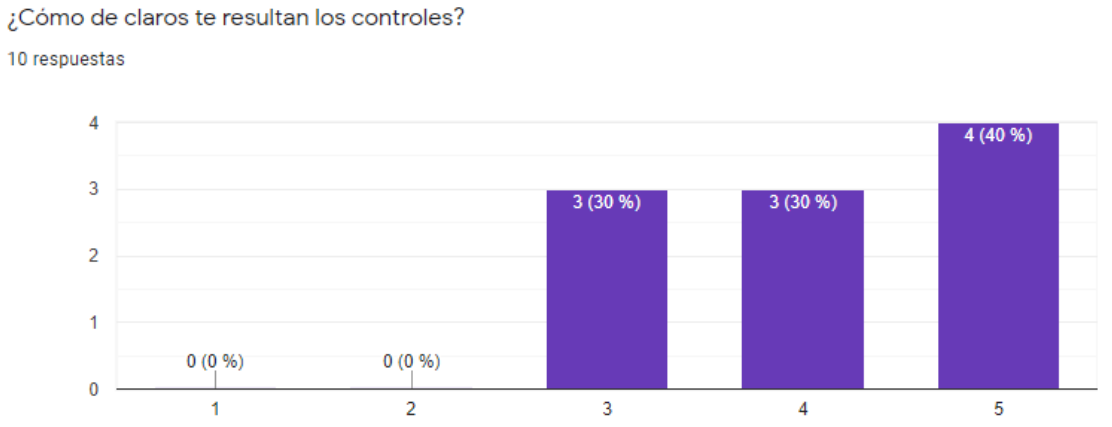
Figura 30: Últimos usuarios en el módulo autenticación

6. Fase de pruebas

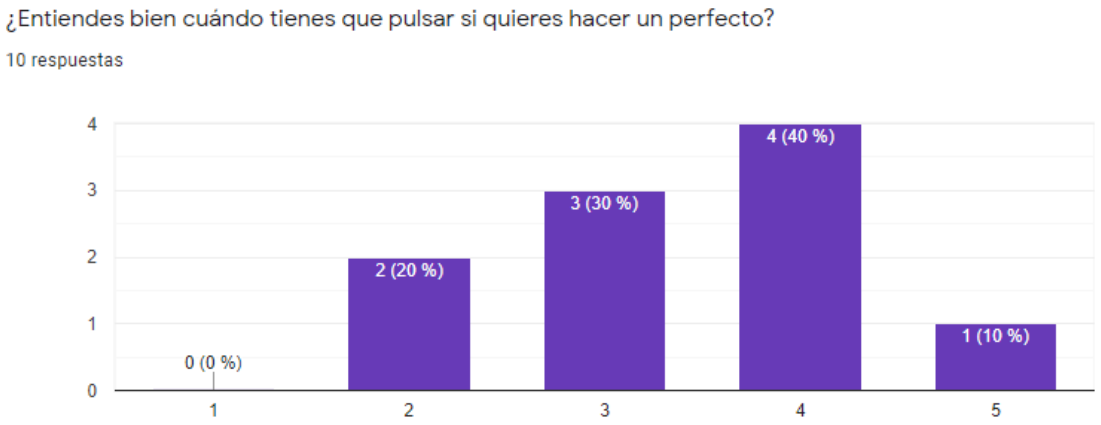
Durante el mes de mayo sacamos una versión para móvil que el usuario pudiese probar con el fin de obtener feedback y detectar posibles errores, por ejemplo a la hora de ver cómo funciona el juego con varios jugadores concurrentes. Para plantear las encuestas usamos un modelo basado en las escalas de Lickert (Joshi et al., 2015, 397-402).

6.1 Implementación Beta

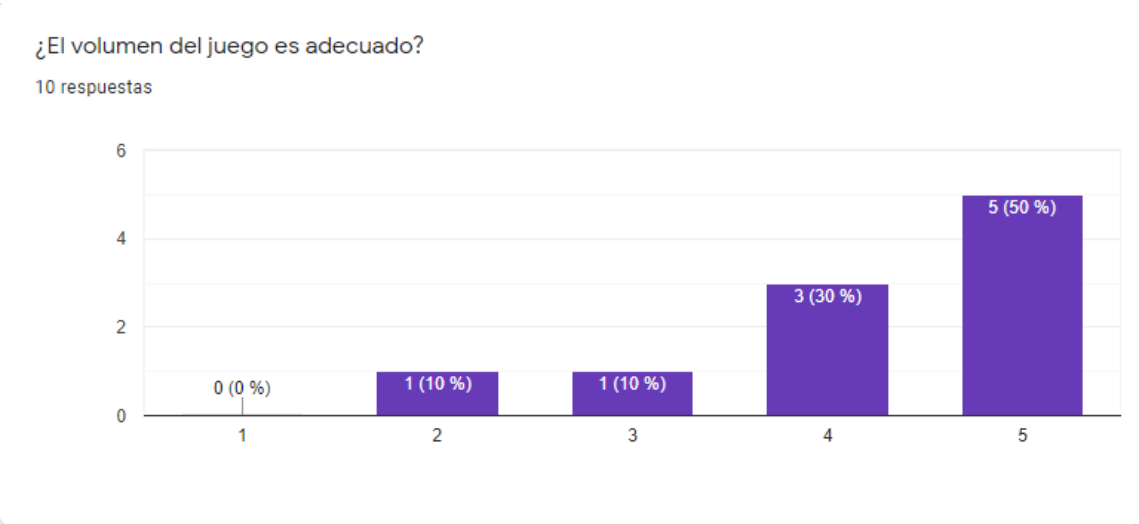
Realizamos una serie de preguntas que los usuarios deben evaluar entre 1 y 5 siendo 1 el peor valor y 5 el mejor valor, además de proporcionar un campo de texto donde podían destacar errores que hayan tenido o si existiese algo que les resultase confuso.



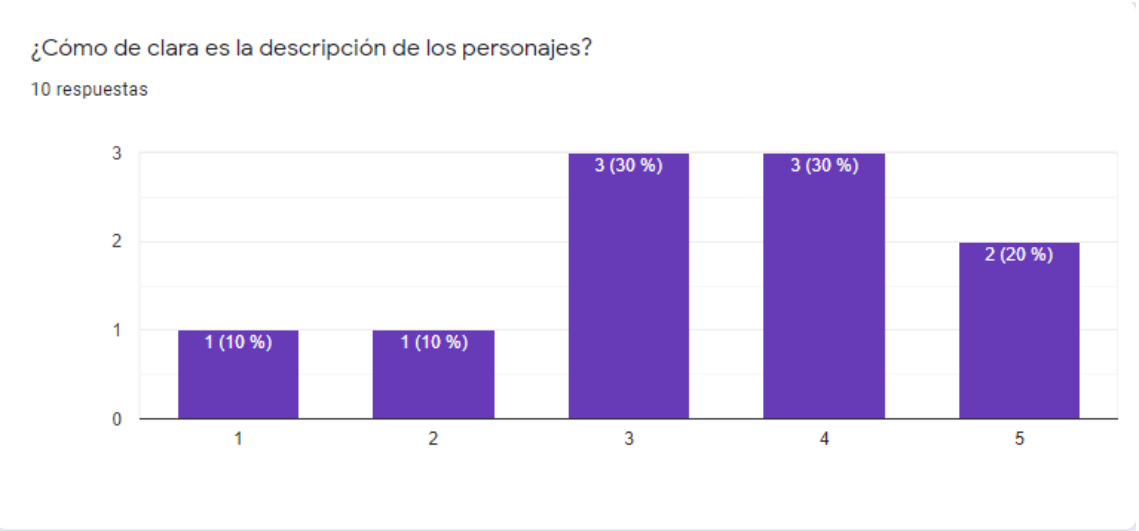
La primera pregunta fue "*¿Cómo de claros te resultan los controles?*". En el gráfico superior observamos una respuesta positiva al manejo de los controles dentro de la partida, por lo que decidimos mantener el diseño.



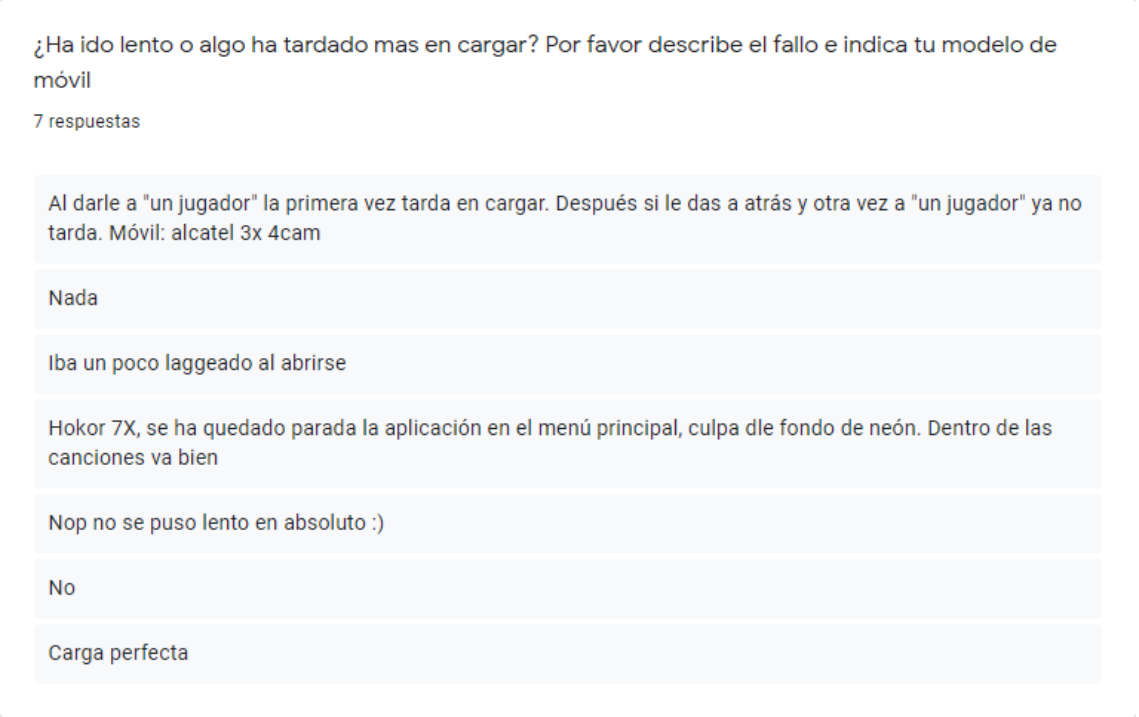
La segunda pregunta fue "*¿Entiendes bien cuándo tienes que pulsar si quieres hacer un perfecto*?". De la respuesta de este gráfico superior deducimos que no todo el mundo comprendió qué es un "perfecto". Esto es debido a que pusimos como personaje por defecto a Topita, un personaje que no posee perfectos.



La siguiente pregunta fue "*¿El volumen del juego es adecuado*?". Para la mayoría de usuarios el volumen del juego es adecuado, pero aún así al ver estas respuestas comprobamos los decibelios de cada sonido de la aplicación.



La última pregunta con formato Lickert fue "*¿Cómo de clara es la descripción de los personajes?*". En este gráfico podemos observar una clara respuesta negativa, cuya solución fue cambiar la descripción de los personajes haciéndola más clara y concisa.



Por último, una de las preguntas de libre respuesta formulaba: "*¿Ha ido lento o algo ha tardado más en cargar? Por favor, describe el nombre del fallo e indica tu modelo de móvil*". Vemos en las opiniones un factor común: el rendimiento en la carga inicial de la aplicación. Como consecuencia hacemos comprobaciones en distintos móviles para observar que, efectivamente, la carga era más lenta.

6.2 Implementación Final

Gracias a los resultados obtenidos tenemos una información con la que realizamos una serie de modificaciones para mejorar la experiencia del usuario:

- Los controles de la pantalla de juego, además de la forma de la nota en cuestión, usan un color distinto para cada nota.
- El volumen del juego en una canción era notablemente distinto de las otras canciones, razón por la cual algunos usuarios indicaron que el volumen era más bajo. Normalizamos los volúmenes de las canciones para evitar estas diferencias.
- Adaptamos las descripciones para que una persona que nunca haya jugado un videojuego de ritmo le fuese más sencillo entenderlo.
- Hubo problemas de rendimiento debido a la carga de recursos en tiempo de ejecución. Lo solucionamos precargando los archivos mp3, usando campos públicos para el referenciado de objetos en Unity y cambiando el fondo a un efecto de partículas de Unity, mucho más ligero en este caso que el vídeo que teníamos originalmente.

7. Trabajo personal

En esta sección explicaremos cómo distribuimos las horas de trabajo para realizar este TFG, cómo nos organizamos y qué hicimos cada uno.

7.1 Planificación del trabajo

Para comprender correctamente el trabajo personal y como se asignó tenemos que repasar la planificación del trabajo previo que hicimos. En esta sección mostraremos cómo organizamos el trabajo con el uso de unos registros de trabajos en línea.

7.1.1 Organización del equipo

Al comienzo del proyecto acordamos reportar las horas trabajadas en un registro de trabajo en línea. Asignamos unas fechas estimadas respecto a las tareas del plan de trabajo. También pactamos unas reuniones semanales, donde fijamos que realizaría cada uno en esa semana y planteamos nuestros problemas semanales respecto a la tarea asignada de la semana anterior. Durante el desarrollo usamos un tipo de desarrollo circular.

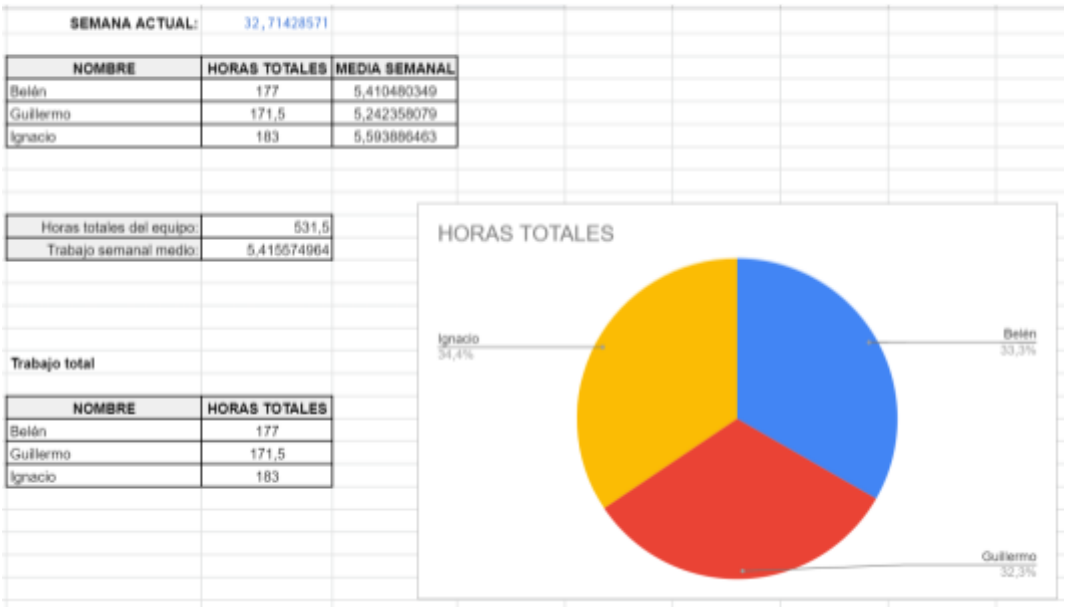
7.1.2 Worklogs

Los *worklogs* son una forma de mantener el equipo motivado y constante en el trabajo semanal. Consisten en una hoja de cálculo en la que cada persona debe introducir una estimación de las horas trabajadas por cada tarea, para después analizar posibles fallos en el rendimiento del equipo o en los objetivos semanales. Por ejemplo, si tras estimar que una tarea se completaría en una semana y al final de la misma se ve que no solo no se ha completado sino que ha llevado más tiempo

de lo esperado, podemos usar estos datos de tiempo y rendimiento para planificar mejor los siguientes *sprints* (Pries & Quigley, 2010) y proponer objetivos más realistas.

La figura 30 que se muestra a continuación contiene la primera página del registro de trabajo. De esta forma podemos observar cuánto ha trabajado el equipo y el trabajo individual de un miembro del grupo de un vistazo rápido. En primer lugar, en la parte superior izquierda y debajo del número de la semana actual del desarrollo, se encuentran los registros de horas totales y semanales por persona. Debajo de estos datos encontraremos otra pequeña tabla con las horas totales del equipo así como el trabajo semanal medio. Finalmente cuenta con un gráfico que ofrece una representación visual de dichas cifras.

Figura 30: vista general de los "work logs"



Las páginas personales (ver figura 31) indican qué hizo cada miembro del grupo organizado por semanas. Así queda registrado el trabajo semanal de cada miembro, además de dar una idea al resto del grupo de en qué se está trabajando y durante cuánto tiempo.

Figura 31: vista semanal de los "work logs"

1		Belén		total:
2			horas (en decimal usando coma)	177
3	semana 1		total semanal:	3
4	8-10-2020	crear un proyecto base en unity (mockup de concepto)	2	
5	9-10-2020	añadir más código al proyecto mockup (spawner y funcionalidad del button)	1	
6				
7	semana 2		total semanal:	2
8	15-10-2020	refrescar los managers para juegos de ritmo	0,5	
9	17-10-2020	meter el manager en el juego, hacer que los cubos spawneen al bpm	1	
10	18-10-2020	setup git en el portátil (nuevo), hacer repo y subirlo	0,5	
11				
12	semana 3		total semanal:	3,25
13	23-10-2020	los botones viajan por la pantalla a velocidad que depende del bpm	0,5	
14		discutir con el equipo cómo escribir el txt de spawner	0,25	
15		añadido score al proyecto, que suma cuando destruyes un cubito	0,25	
16	24-10-2020	comentar UI con Guille	1	
17	25-10-2020	mejora de la detección del player input. Dos formas-> o con el teclado o dando a un botón en la parte inferior (esto, pensado para móviles). Ligera reestructuración del código: GameManager, y clase base con funcionalidad compartida para los botones	1,25	

7.1.3 Bocetos y planes originales

Antes de que se unieran el resto de compañeros, el proyecto estaba planteado para una sola persona (Belén Solla), que acordó con el coordinador que el juego se haría de cero usando un motor de videojuegos. Nos decidimos por usar Unreal, seguido muy de cerca por Unity. La razón por la que usar Unreal al final se reducía al aprendizaje personal junto con el prestigio que tiene Unreal como motor de videojuegos.

Cuando llegaron el resto de los compañeros, al no tener experiencia con ningún motor de videojuegos, hubo un ligero cambio de planes y decidimos usar Unity como motor. Esto es debido a que tiene una curva de aprendizaje más asequible que Unreal, además de un gran número de tutoriales y otros recursos fácilmente disponibles. No comprometimos la calidad del proyecto, ya que aunque es cierto que Unreal es superior en cuanto a fotorealismo, nuestro juego tiene un estilo más arcade y simple que se adapta perfectamente a los requerimientos de Unity. Además, acordamos utilizar Github como software de control de versiones.

En las primeras semanas del trabajo en el TFG discutimos cuál sería la estética del programa y el diseño de **Tamb-On**. De forma paralela creamos un mockup del juego en Unity con mecánicas muy simples, a la vez que diseñamos unos bocetos utilizando Balsamiq Cloud (Balsamiq Studios, n.d.).

7.2 Guillermo Alonso

Mi historia en este TFG comienza en septiembre, reuniendome con mi compañero y amigo del curso, Ignacio, para elegir uno de los TFGS ofrecidos por la universidad. Vimos varias propuestas interesantes, pero claramente destacó entre las demás el desarrollo de un videojuego rítmico, teníamos un interés común en este tema y por lo tanto la elección estaba clara.

En las primeras semanas de octubre fui aprendiendo Unity creando un pequeño proyecto, un juego, para familiarizarme con el nuevo entorno, mientras aprendía con documentación y videos recomendados por Belén que eran bastante útiles. Al comienzo de este periodo no tenía conocimientos de Unity, por eso agradezco a Belén su ayuda para superar este obstáculo. También mejoré el registro de trabajo, corrigiendo los fallos en algunas fórmulas, rematando otras y creando una gráfica en función del número total de horas trabajadas por cada participante. Así, de un vistazo, podíamos ver cuánto hemos trabajado cada uno rápidamente.

Mi primera contribución al proyecto ocurrió a mediados de octubre, en paralelo con mi aprendizaje en Unity: procedí a registrar una cuenta común para el proyecto en la página Balsamiq y cree los mockups de los menús del juego. Estos eran muy simples pero servían como boceto y un primer diseño básico. Balsamiq permite la creación de diapositivas con botones interactuables que permiten dar una idea de cómo sería la funcionalidades del menú. En estos diseños se notaba mi inexperiencia en la creación de videojuegos y Belén me fue guiando para mejorar mi diseño de la IU.

Proseguí con la creación de los menús a finales de octubre, aunque aprendí cómo crearlos y lograr que la funcionalidades del menú fueran las esperadas, realmente no conseguía dejarlo con una estética agradable, por lo que cedí la finalización de los menús a Ignacio para centrarme en el desarrollo del sistema de puntuaciones.

Comencé la implementación del sistema de puntuaciones a mediados de noviembre, añadiendo el campo “Score”, que aparece en la esquina superior derecha de la escena de juego y una sistema de puntuación básica asociada a él. La escena de juego que utilice fue creada por Belén, este sistema tenía una puntuación plana donde todas las notas puntúan por igual. La lógica del sistema era que cada vez que el jugador pulsaba un botón de golpeo de nota este botón llamaba a un método codificado en C#. Este método fue creado y añadido al *InputManager.cs*, explicado en la sección 5.7.1. Por otro lado, la puntuación fue agregada a *ScoreManager.cs*, explicado en la sección 5.7.1. La lógica de la puntuación implementada en esta fase era muy simple, simplemente agregaba la puntuación correspondiente según si era un perfecto, bien o fallo. El mayor problema que tuve en este punto fue cómo implementar las colisiones de las notas con las barras de golpeo, las cuales las añadí en este punto, y cómo interactúan entre sí con el usuario, una mecánica básica del juego. El primer diseño de este sistema contenía tres barras, siendo la barra central la que representaba el

perfecto y las barras laterales los aciertos. Estas barras en cuestión se podían solapar con las notas de la melodía y si el jugador pulsaba el botón adecuado para cada nota entonces se llamaba a una función que comprobaba el solapamiento con las barras, si se solapa, entonces se contaría como acierto o si la nota no se solapaba entonces se contaría como un fallo. El primer problema que me ocurrió con este sistema era qué ocurriría si una nota se solapaba con dos barras. Para solventar el problema, dejé un poco menos del espacio justo de una nota entre cada barra. Así, me aseguraba que siempre solapará con una barra si estaba entremedias de estas. Otro de los problemas iniciales con este sistema es que al comienzo de estas comprobaciones se realizaban de forma incorrecta, priorizando las barras de acierto sobre la barra de perfecto, un fallo tonto podriamos decir, la solución fue sencilla: corregir el orden.

A comienzos de diciembre consideramos que cambiar a Unity Teams sería un acierto y dejamos atrás Github.

Una vez creada la parte básica del sistema de puntuación continué con la creación de los combos, la segunda semana de diciembre, justo después de migrar a *Unity Teams*. En este punto tuvimos varias ideas de cómo podría ser. La primera idea fue un sistema de combos clásico, donde cada nota acertada de forma consecutiva puntuará más. En particular, teníamos dos variantes de esta idea: por un lado que existiera un incremento positivo y que este fuese lineal o que fuese exponencial. La segunda idea era un combo que en lugar de ser positivo fuese negativo, es decir, si se fallaba varias notas consecutivas se restarían puntos. La idea implementada fue la primera en su variante exponencial. Sin embargo, debido a la división de opiniones al respecto, surgió la idea de trasladar estas opciones a algo que pudiese elegir o configurar el jugador. De esta forma todos tendrán el modo de juego que querían. Así surgió la idea para los mutadores de *precisión* y *combo*, como los personajes de *Caxper* y *Topita.*

Ignacio estaba sobrecargado a finales de diciembre, cuando terminé de crear el sistema de combos, así que me propuso que investigara sobre un carrusel para la escena de selección de canción y lo implementara. El resultado final tenía una gran pega. El slider se descuadraba si existían más de un número determinado de canciones en la lista. Como la idea era que pudiese soportar infinitos elementos sin que se descuadre optamos por descartar este diseño.

En este punto, en la primera semana de enero, decidimos centrarnos en los exámenes y trabajos, dándonos una pausa del TFG hasta febrero.

A mediados de febrero comencé la redacción y estructuración de la memoria, centrándome sobre todo en investigar memorias de otros TFGs.

Una semana después ya casi teníamos una versión *alpha* del juego, teníamos la IU, las mecánicas básicas y teníamos un sistema para introducir canciones al juego, pero solo teníamos una canción creada por lo que decidimos que era buena idea crear dos canciones más, yo me dedique a crear una de estas dos canciones.

Aproveche esta creación y la introducción de la misma al juego para crear una guía para los usuarios, que documenta todo el proceso.

Una vez tuvimos lo básico para jugar quedaba agregar unas mecánicas para que el juego fuese más divertido y estuviese listo para la alpha, por lo que mi siguiente cometido fue el desarrollo de los personajes y mutadores con Ignacio, nos pusimos a ello a mediados de marzo, agregando el código correspondiente a ScoreManager.cs, sección 5.7.1. También creé una guía de uso por si acaso alguien la necesitaba en la fase de pruebas. Sin embargo, nadie la necesitó y esta guía quedó desactualizada en versiones futuras y fue abandonada.

Con la versión alpha lista en abril, procedimos a comenzar la fase de pruebas, pasando el juego a nuestros contactos para que lo probaran. Mientras fui redactando la memoria del TFG. Después de las pruebas sacamos algunos errores que a nosotros se nos pasaron, lo cual tiene bastante sentido pues el juego lo desarrollamos nosotros. Corregi los errores con el resto grupo a mitad de abril, la mayoría de los fallos eran en la IU, donde algunas cosas no resultaban tan intuitivas como creíamos, los usuarios se quejaban de que no sabían cuándo acertaban o hacían un perfecto, también se quejaron de que algunas descripciones no eran lo suficientemente claras. Otros errores ya sabíamos de su existencia: en concreto, arreglé un error en el sistema de puntuación que provocaba que Jerry, el nombre provisional en ese momento de Caxper, no restaba correctamente los combos negativos. Un problema que me dio que fue especialmente difícil para mi fue el rendimiento del juego, como nunca desarrolle uno este problema era nuevo para mi, aprendí bastante de Belén en este punto, pues no sabía cómo instanciar los objetos Unity y teníamos bastantes cargas innecesarias.

Después de eso me dediqué a desarrollar la memoria del TFG e investigar la normativa y otras memorias para saber cómo hacerla.

7.3 Ignacio de la Cruz

Al igual que Guillermo, comencé este TFG haciendo un análisis del listado disponible. De entre los muchos proyectos que nos parecían plausibles, este proyecto satisfacía mi interés por hacer un proyecto relacionado con la informática musical, y además cumplía el de Guillermo de tener la oportunidad de hacer un videojuego. Como extra contamos también con Belén, estudiante del Grado de Desarrollo de Videojuegos que cuenta con amplia experiencia en este tipo de proyectos.

Al inicio del proyecto, sobre octubre de 2020 lo primero que hicimos fue plantear una planificación de cómo desarrollaríamos el juego, inicialmente con un diagrama de Gantt, sin embargo al final optamos por los *worklogs* previamente explicados dado que daban una representación visual más clara sobre cómo trabaja cada

miembro del grupo. También pusimos fechas límites para ciertos aspectos del desarrollo, queríamos tener un juego probable para finales de año, para poder pulirlo y tener una versión distribuible sobre marzo o abril del año siguiente para poder recibir opiniones de distintos usuarios, nuestra intención es que el juego sea entretenido y que cualquier usuario quiera volver a jugar.

Los primeros pasos que dimos en el desarrollo fueron los necesarios para el aprendizaje del motor y del lenguaje de programación C#. El lenguaje no me resultó nada complicado dado que es muy parecido a Java, lenguaje que uso a nivel profesional. Para poder comprender cómo usar Unity leí su documentación (Unity Technologies, 2021), vi tutoriales básicos y probé a programar los juegos básicos que se incluyen dentro del motor. En unas semanas conseguí dominar los aspectos más importantes de la programación de un videojuego 2D en Unity.

Estudié la posibilidad de poder implementar un modo multijugador, mi siguiente tarea fue averiguar la forma de poder implementarlo sin consumir recursos excesivos, es importante para nosotros que no exista una latencia inconsistente, es algo que nos resulta muy molesto cuando nosotros jugamos a un videojuego. Toda esta información obtenida fue descartada cuando más adelante decidimos usar Firebase para el almacenamiento de datos y se vió la posibilidad de poder usarlo además para un modo multijugador.

A estas alturas del proyecto, aproximadamente mediados de noviembre, Belén ya disponía de un juego que desplazase un elemento de derecha a izquierda y este fuese destruido cuando entrase en colisión con una malla al mismo tiempo que fuese pulsada una tecla, lo único que faltaba era determinar la frecuencia de aparición de estas notas, aquí es donde entra en juego el parseo de un fichero .mid. Lo primero que hice fue crear un fichero MIDI de ejemplo con distintas notas de distintas duraciones, más adelante realizaría la traducción de una canción real. Para poder ver en qué instante de tiempo una nota debía aparecer descubrí que tan solo era necesario un cálculo matemático que incluía el tiempo relativo del MIDI y el número de nota, accesibles en formato hexadecimal en el fichero y el bpm al que se reproduce la canción, junto con ciertas constantes, de esta forma podía determinar que tipo de nota era (arriba izquierda, abajo derecha, etc.) y el instante de tiempo en el que debía aparecer, sumándole además el tiempo en el que tarda en recorrer desde el lugar en el que aparece hasta que llega a la malla.

Después de esto me dediqué a rediseñar la interfaz de usuario del menú que sufriría más cambios en el futuro, la razón principal de esto fue la estética, necesitábamos un estilo uniforme para todos los aspectos y no nos poníamos de acuerdo.

Seguidamente a finales de diciembre entra en juego la base de datos y el almacenamiento de información, con un poco de investigación concluí que la tecnología más factible era la de Firebase, dada su fácil integración con unity y amplias características. El primer módulo que desarrollé fue el de “Records” que consistía en una escritura cuando se acababa una canción, y una lectura en caso de

entrar en la pantalla de visualización. En este punto del desarrollo decidimos que yo haría la funcionalidad de la interfaz de usuario y Belén su estética.

Por último, antes del parón que sufrimos debido a los exámenes de enero, tanto Guillermo como yo nos dedicamos a hacer traducciones MIDI de canciones reales, usando Reaper. Usamos canciones de libre uso que nos parecieron que eran apropiadas para un juego de este tipo, es decir, que tuviesen un ritmo muy marcado para que fuese obvio cuando sería lógico insertar una nota. El proceso manual de la creación del fichero es tedioso y repetitivo, sin embargo bastante fácil de comprender. En reaper al insertar un archivo de sonido se puede ver la onda de sonido que genera, generalmente se quiere insertar una nota en los momentos en el que la amplitud de esta onda es mayor, por ejemplo en la canción de Monody se puede ver que la onda tiene una amplitud mayor cuando la cantante hace énfasis en la acentuación de una palabra.

Cumplimos los plazos que nos habíamos establecido, por lo que en la segunda parte de este desarrollo nos dedicamos a pulir los aspectos que habíamos implementado. Personalmente tuve dificultades con la base de datos y el resto de módulos de firebase que acababa de implementar, en general suelen resultar complicaciones cuando se hace comunicación externa por internet o con otros usuarios, principalmente en el uso concurrente de la aplicación, hicimos mejoras a las canciones, colocando las notas en sitios más precisos.

Aprendí cómo implementar el uso de un mando de Wii dentro del juego, siendo capaz de detectar la pulsación de un botón de dicho mando además de poder ver a tiempo real la posición relativa en tres dimensiones del mando además de su rotación, aunque finalmente descartamos esta idea.

En abril y mayo llegamos a la fase de pruebas, donde preguntamos a compañeros de la carrera sus opiniones sobre los temas que más nos conciernen, como era el rendimiento de la aplicación o sugerencias para que el juego resultase más ameno. Un objetivo personal es ser capaces de desarrollar un juego entretenido en la medida que se pueda, además de aprender sobre el procesamiento de las ondas musicales en la informática.

Por último los tres miembros del equipo nos dedicamos a redactar esta memoria, cada uno se implicó más en la redacción de las partes en las que estuvo más involucrado, por ejemplo como fui yo quien desarrolló la base de datos, durante el proceso fui documentándolo para poder plasmarlo en esta memoria como se puede observar en el punto 5.8. También documenté una gran parte de la sección 5 puesto que estuve muy relacionado con el desarrollo

7.4 Belén Solla

A la hora de elegir el trabajo de fin de grado, este me llamó la atención porque la temática estaba basada en los videojuegos. Ya que soy estudiante del Grado de Desarrollo de Videojuegos, sentí que podía ser una buena oportunidad y me decidí por apuntarme. Al principio solo estaba yo en el proyecto así que empecé a investigar y a aprender Unreal Engine, el motor que pensábamos usar en un principio. También leí numerosos blogs y foros sobre cómo crear juegos de ritmo, en los que encontré consejos que resultaron bastante útiles durante el desarrollo del juego.

Cuando el grupo se amplió me dediqué a crear un prototipo en Unity mientras mis compañeros aprendían las bases del motor. En este prototipo implementé lo que serían las bases de nuestro juego: el comportamiento base de los botones, el *spawner* y algunos *managers*. Su funcionamiento era muy limitado todavía, ya que estábamos en una fase muy temprana del desarrollo. Sin embargo continué añadiendo funcionalidades a este proyecto, que luego se convertiría en el proyecto que tenemos actualmente. El primer mes fue casi exclusivamente de prototipado para mí, mientras mis compañeros conseguían un nivel de Unity suficiente para el trabajo.

Durante el segundo mes mis compañeros empezaron a programar ya dentro del proyecto, así que empecé a trabajar en algunos temas de arte. En ese momento no teníamos muy clara la estética final que queríamos para el juego, así que lo que pensé fue en hacer un modelo 3D de un tambor taiko y usarlo de "botones" pulsando sobre cada zona del tambor. Como se mencionó anteriormente, este modelo no se usa en la versión actual ya que habíamos encontrado una estética que se adecuaba más al proyecto, usando exclusivamente texturas 2D.

En diciembre volví a tomar un rol más centrado en la programación y me dediqué a retocar algunos *scripts* que habían hecho mis compañeros y a conectar todo correctamente mediante el Game Manager. Las reuniones de equipo eran bastante productivas ya que los compañeros tenían ganas de aprender y estaban abiertos a cambiar ciertos aspectos de su código. En este punto del desarrollo empezamos a iterar sobre el aspecto visual. Cambié ciertas configuraciones relativas al renderizado y la luz que daban problemas de artefactos visuales. Creé las primeras texturas para los botones, que posteriormente serían revisados, como explicaré más adelante. Durante este mes nos reunimos bastante frecuentemente, ya que había muchos asuntos que discutir y dudas que resolver.

A mediados de mes, buscando una solución a los problemas que teníamos con la biblioteca de procesado de MIDI *SMFLite*, incorporé al proyecto la biblioteca *DryWetMidi*. El problema era sobre todo que en un principio las notas se leían y creaban simultáneamente, lo que no era suficientemente preciso para un juego de ritmo como el nuestro. Al incorporar la nueva biblioteca, cambié la forma en que se procesaban los MIDIs para que las notas se leyeran primero y se almacenaran en una cola, de la que luego leería el *spawner*. Además, añadí un sistema de *flags* para

el manejado de las variables que indicaban si una nota estaba dentro de las zonas de golpe, condensando así numerosas variables booleanas en un solo *flag*. Así, para comprobar si uno de estos *flags* estaba activado, se hacía una comprobación a nivel de bits usando operaciones lógicas. Por último empecé a planear los "botones especiales", empezando por la nota de *drumroll* o tamborileo. En un principio queríamos que el juego tuviera más tipos de notas: las cuatro notas que están en el proyecto final y las notas especiales de tamborileo y golpe sostenido, que no llegaron al juego final.

Enero fue bastante escaso en cuanto al desarrollo del juego ya que estábamos ocupados estudiando y haciendo exámenes, así que lo único que incorporé fue una versión más pulida de la nota de tamborileo. A finales de febrero retomamos de nuevo el proyecto, y empecé a trabajar en la nota de golpe sostenido. Desarrollé clases para estas notas especiales, con dos tipos por cada nota especial (una para los botones "centrales" del input y otra para los "exteriores"). La razón por la que decidimos no incluir las notas especiales en el juego final es que decidimos que dificultaban el juego en exceso, o lo hacían confuso ya que se comportaban de forma tan distinta a las notas normales. El código de estas notas especiales todavía se encuentra en el proyecto, para dar la oportunidad de arreglar esos errores de diseño en iteraciones posteriores del proyecto.

Marzo fue dedicado sobre todo al desarrollo del arte para los personajes, aunque también programé algunas funcionalidades dentro de los módulos de *Input* y *Spawner*. En este punto del desarrollo ya teníamos la funcionalidad de los personajes, pero estos tenían todavía imágenes provisionales por lo que era el momento de centrarse en el lado visual del juego. Para ello nos reunimos de forma periódica para comentar los bocetos e ideas que iba creando para los personajes. Conseguí acabar los cinco personajes en un periodo de unas dos semanas, usando Krita para el proceso de dibujo digital como se menciona en la sección 5.6, *Diseño y Arte de Mutadores y Personajes*.

En abril seguí con el desarrollo de texturas y arte para el juego, centradas esta vez en elementos de la IU (interfaz de usuario). Decidí crear una serie de texturas vectoriales en Krita, y luego ajustar correctamente la imagen importada en Unity para que, al escalarla, solo se deformara el centro del botón. Esto nos permitió reutilizar el arte para distintos botones, ya que aunque usáramos la misma textura para un botón grande y uno pequeño, la configuración aseguraba que los bordes y forma general del botón se mantenían. Durante esta fase también se empezaron a pulir los diseños y la colocación de los elementos de la IU para cada menú. Ejemplos de texturas para el menú podrían ser los botones de retroceso, las texturas de los elementos de la tabla de récords , o las texturas de los botones de elección de canción, entre otros (ver figura 8 en la sección 4.1.4, *Visualización de récords* y figura 6 en la sección 4.1.2, *Navegación por los menús, selección de canción*). También empecé a crear las texturas para los mutadores y a incorporarlas al juego, además de incorporar nuevas texturas para los botones de la escena de juego.

Queríamos tener el proyecto listo en mayo para poder hacer pruebas con usuarios e iterar sobre él antes de la entrega en junio, así durante este mes me centré en terminar el resto de texturas que faltaran, además de crear efectos de partículas para el fondo y el efecto que aparece al golpear una nota. También guié al equipo a la hora de hacer las pruebas con usuarios, ya que yo tenía experiencia con este tema. Cuando tuvimos un cuestionario preparado, lo distribuimos y tomamos nota de la reacción de los jugadores para hacer los últimos cambios e iteraciones al proyecto. Uno de esos cambios consistió en cambiar por última vez la textura de los botones de juego.

Este mes de junio lo dedicamos casi exclusivamente a preparar la memoria. Me encargué de preparar un repositorio público con el código del proyecto final, además de una página muy sencilla donde descargar la aplicación en formato *apk*. En cuanto a la escritura de la memoria, me encargué de las partes referentes a las descripciones de mecánicas y dinámicas, las secciones relacionadas con el arte y la traducción de las partes escritas en inglés. También me encargué de recopilar la bibliografía necesaria para el proyecto. Durante este mes nos reunimos frecuentemente para editar el documento en equipo, así que también he contribuido en menor medida en otras secciones a lo largo de la memoria.

Creo que gracias a mi experiencia y educación previa, he conseguido tener un buen rendimiento en este proyecto. Ha sido enriquecedor poder hacer de guía para mis compañeros en áreas relacionadas con los videojuegos, y por supuesto también he aprendido de ellos. Durante todo el desarrollo traté de seguir las guías que nos habían propuesto profesores de distintas asignaturas, y sobre todo tomé mucha inspiración de mi experiencia en el Erasmus donde desarrollamos varios juegos. Este proyecto para mí es un testimonio de que merece la pena ser constante y planificar el desarrollo para conseguir una buena dinámica de equipo y un producto final satisfactorio.

8. Conclusiones y Trabajo a futuro

Los objetivos propuestos para este TFG se cumplieron satisfactoriamente. Se creó un juego similar a Taiko no Tatsujin, que da la oportunidad al jugador de personalizar sus niveles tanto en configuración, como en dificultad del propio nivel. Además, permite la creación de niveles con canciones que se basen en el estándar MIDI. Adicionalmente, se implementó un sistema de amigos que permite ver qué amigos están en línea y cuáles están jugando, si están jugando alguna canción. También se implementó un registro de récords en línea. Esto permite al usuario consultar sus récords o el de sus amigos siempre que tenga conexión a internet.

La experiencia obtenida del trabajo realizado es bastante gratificante. Aprendimos tanto de Unity, como de diseño y desarrollo de videojuegos. Además, también cumplimos con gran satisfacción este cometido, pues nos gustan los videojuegos y su desarrollo. El trabajo de equipo fue equilibrado y el trato cordial.

La integración del sistema MIDI se completó en las primeras fases del desarrollo, después de los diseños de los Mockups. Como MIDI es un estándar popular implica que muchas canciones tendrán su archivo MIDI para que el jugador pueda incluirlo en el juego o que haga él mismo su propio MIDI a su gusto. Además, la creación de los propios niveles es bastante precisa para los jugadores más perfeccionistas.

El juego es bastante divertido y variado. Contiene múltiples opciones combinables que alargan la vida útil del mismo. El equilibrio dentro del juego es adecuado a la dificultad del desafío, siendo este desafío configurable gracias a los mutadores. Los personajes agregan una variedad adicional a la anterior mencionada, siendo esta combinable con los mutadores. El sistema de puntuación puntúa en función del grado de precisión, si un jugador atina con una precisión milimétrica logra un perfecto mientras que si atina aproximadamente lograra un acierto estándar.

El director de este TFG siempre logró un ambiente amigable y cordial con el equipo y estaba disponible para resolver cualquier duda. Por nuestra parte fue un placer trabajar con él.

Teniendo en cuenta lo anterior, el proyecto está completado.

8.1 Trabajo a futuro

A continuación describiremos características que se plantearon pero dada su complejidad o falta de recursos decidimos posponer.

- Una sección donde el usuario pueda crear sus propias canciones. La idea principal consiste en que el usuario suba el MP3 a la aplicación y pueda

moverse con total libertad hacia adelante y atrás en el tiempo para colocar las notas donde desee. Finalmente esta información se traduce por código a instrucciones *MIDI* y finalmente se sube un archivo *.mid* a la nube en Firebase para su posterior uso.

- Un modo multijugador en el que cada usuario registrado puede crear o unirse a una sala privada (con contraseña) o pública (sin contraseña). El usuario que cree la sala actúa como anfitrión y marca tanto el comienzo de la partida como la canción que se va a jugar. Está pensado para un máximo de 8 jugadores por sala y además durante el juego se ve en tiempo real una clasificación que indique la puntuación de cada jugador.
- La posibilidad de jugar con un mando de consola estándar, o con un mando de *Wii*, por conexión *bluetooth* al móvil. Para elegir qué nota se está pulsando se deberá orientar el mando de *wii* relativamente a una posición inicial calibrada, además de pulsar una tecla para marcar el momento de pulsación de la nota.
- Incluir el diseño artístico de los personajes dentro de la propia partida junto con una pequeña animación de movimiento.
- Distintos modos de juego.
- Distintos tipos de nota, como una donde el jugador deba mantener pulsado el botón durante un determinado tiempo u otra similar donde el jugador deba pulsar repetidamente un número determinado de veces.

9. Conclusion and future work

The goals set for this TFG were amply achieved. A similar game to Taiko no Tatsujin was made, and it provided the player with the option to customize levels. This was not only difficulty wise, but also mechanics wise thanks to mutators and characters. It also allows for MIDI-based level creation. A friend system was implemented as well. It allows players to see which friends are online and which ones are currently playing any songs. A leaderboard was also implemented. The player can check their own best scores, as well as a global best ranking, provided they have an internet connection.

Experience gained from our work in the project was quite rewarding. We learned both how to use Unity Engine and about video game design and development. We felt fulfilled as we worked on the project as well, because video games and their development were topics of interest to us. Teamwork was balanced and the atmosphere was friendly.

The integration of the MIDI system was completed in the early stages of development, right after the game mockups. MIDI is a popular standard, so that facilitates finding songs in MIDI format or even the creation of custom MIDI levels by the player. Thanks to MIDI's very fine precision, level creation is made easier for those players looking for excellent accuracy.

The game is quite fun and varied. It contains multiple settings that can be combined with each other to extend its useful life. Balance within the game is appropriate to the difficulty of the challenge and configurable thanks to the mutator system. The score logic takes into account player performance: an accurate beat hit will yield a "perfect" and a less accurate hit will result in a regular "okay" hit.

The director of this TFG always achieved a friendly and cordial atmosphere with the team and was available to answer any questions. On our behalf, it was a pleasure to work with him.

Taking the above into account, the project is successfully completed.

9.1 Future work

The following list below describes features that were proposed, but due to either their complexity or a lack of resources had to be postponed.

- A song creator mode, where the user can create their own levels. The main idea consists of allowing the user to upload an MP3 to the application, then browse the timeline of the song to place beats wherever they choose. Finally, the information produced would be codified into a MIDI file and sent to the Firebase cloud for later use.
- Multiplayer mode: any registered user can create or join either a private (password-protected) or public (free-access) lobby. The user that created the lobby is the host and controls both game start and song to be played. It is designed for a maximum of 8 players per lobby. A real time high score classification will also appear during the song, showing the current best score of each player.
- Optional control schemes using a standard console controller or a *Wii* controller connected to the phone by bluetooth. To choose the type of beat the player would point the controller in a certain direction depending on an initial calibration. Then, when the beat is over the goal, press a button to hit it.
- Add the characters into the play scene and add some animations to them.
- Different game modes.
- Different types of note, such as one where the player must hold down the button for a certain time or a similar one where the player must repeatedly press a certain number of times.

10. Bibliografía

Balsamiq Studios. (n.d.). *Balsamiq for Desktop Documentation*. Balsamiq for Desktop Documentation | Balsamiq. https://balsamiq.com/wireframes/desktop/docs/

Firebase. (2021, Junio 1). *Firebase Authentication*. Firebase Authentication. https://firebase.google.com/docs/auth?authuser=0

Firebase. (2021, Junio 1). *Firebase Realtime Database*. Firebase Realtime Database. https://firebase.google.com/docs/database?authuser=0

Firebase. (2021, Junio 8). *Add Firebase to your Unity project*. Add Firebase to your Unity project. https://firebase.google.com/docs/unity/setup?authuser=0

Huber, D. M. (2007). *The MIDI Manual: A Practical Guide to MIDI in the Project Studio* (3rd ed.). Focal Press. https://doi.org/10.1016/B978-0-240-80798-0.50003-2.

Joshi, A., Kale, S., Chandel, S., & Pal, D. K. (2015). Likert Scale: Explored and Explained. *British Journal of Applied Science & Technology*, 7(4), 396-403. www.sciencedomain.org

KDE. (2021, Junio 11). *Welcome to the Krita 4.4 Manual!* Welcome to the Krita 4.4 Manual! - Krita 4.4.0 documentation. https://docs.krita.org/en/index.html

Olsson, M. (2013). *C# Quick Syntax Reference* (1st ed.). Apress, Berkeley, CA. https://doi-org.bucm.idm.oclc.org/10.1007/978-1-4302-6281-7

Pries, K. H., & Quigley, J. M. (2010). *Scrum Project Management*. Taylor & Francis Group. https://ebookcentral.proquest.com

Reaper. (2021, Mayo 18). *The REAPER User Guide*. REAPER | User Guide. https://www.reaper.fm/userguide.php

Schell, J. (2014). *The Art of Game Design : A Book of Lenses, Second Edition*. CRC Press LLC. https://ebookcentral.proquest.com/lib/universidadcomplutense-ebooks/reader.action?docID=1598419&ppg=204

Unity Technologies. (2021, Junio 4). *Manual de Unity*. Unity - Manual: Manual de Unity. https://docs.unity3d.com/es/530/Manual/UnityManual.html